ffice of
cademic
omputing

SUPPORTING THE IBM FILE SYSTEM IN NSW

NSW Semi-Annual Technical Reports

January 1, 1978 - June 30, 1978
and
July 1, 1978 - December 31, 1978

UCLA TR-23

Neil Ludlam
Steve Farrell
Robert Braden

niversity of
alifornia,
os
ngeles

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS
BEFORE COMPLETING FORM

1. REPORT NUMBER
OAC/TR23

2. GOVT ACCESSION NO.
ADA186886

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
Supporting the IBM File System
National Software Works

5. TYPE OF REPORT & PERIOD COVERED
Semi Annual Tech Reports
1/1/78 - 12/31/78

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
R. Braden - N. Ludlam - S. Farrell

8. CONTRACT OR GRANT NUMBER(s)
MDA 903-74-C-0083

9. PERFORMING ORGANIZATION NAME AND ADDRESS
University of California
Office of Academic Computing
Los Angeles, CA 90024

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
PROG. ELEMT. 62708E
Program Code: 0T10
ARPA Order No. 2543

11. CONTROLLING OFFICE NAME AND ADDRESS
Defense Advanced Research Projects Agency
Attn: Program Mgt Office
1400 Wilson Blvd., Arlington, VA 22209

12. REPORT DATE
11/20/80

13. NUMBER OF PAGES
152

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
Defense Supply Service- Washington
Room 1D-245, The Pentagon
Washington, D.C. 20310
R. Mueller

15. SECURITY CLASS. (of this report)

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

~~Distribution Unlimited~~

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

National Software Works, NSW, ARPANET, OS MVT, NSW file package, NSW copy machine, NSW Library, FP/360, BCM, configuration management, sequential files, partitioned files, IBM software tools.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report covers technical development at UCLA relating to the National Software Works (NSW) during 1978. It is specifically concerned with the design and implementation of the NSW File Package component under 360 OS/MVT.

SUPPORTING THE IBM FILE SYSTEM IN NSW

by
Neil Ludlam
Steve Farrell
Robert Braden

November 20, 1980

Document TR-23

UCLA Office of Academic Computing
5628 Math Sciences Addition
University of California C0012
Los Angeles, California 90024

This work was sponsored by
the Advanced Research Projects Agency
of the Department of Defense,
under ARPA Order no. 2543,
Contract No. MDA 903-74-C-0083:

ARPANET COMPUTER SERVICES IN SUPPORT OF
THE NATIONAL SOFTWARE WORKS

June 1, 1975 - February 29, 1980

William B. Kehl, Principal Investigator
(213) 825-7511

SEMI-ANNUAL TECHNICAL REPORTS
for period of
January 1, 1978 - December 31, 1978

The views and conclusions contained in this document are those of the authors, and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or of the United States Government.

REPORT SUMMARY

This report covers technical development at UCLA relating to the National Software Works (NSW) during 1978. It is a combination of the two Semi-annual Technical Reports covering the periods of January 1 through June 30 and July 1 through December 31 of 1978.

The primary goal of the NSW project at UCLA is to make the IBM Operating System OS/MVT, and specifically its implementation on the UCLA IBM 360/91, a "tool-bearing host" within the NSW. This report is specifically concerned with the design and implementation of the NSW File Package component under OS/MVT. The next three sections of the report correspond to specific documents stored in the NSW documentation repository maintained by the NSW Operations Contractor, so each section has been made self-contained. For example, each section has its own table of contents and reference summary, and each section is independently paginated.

Part II: FP/360 -- The NSW MVT File Package

This section describes FP/360, the File Package implementation for OS/MVT, from the aspect of its use as an NSW core-system component. It does not go into program logic to any depth.

Part III: The NSW Basic Copy Machine

This section describes that subcomponent of FP/360 called the "Basic Copy Machine", or BCM. The BCM can be viewed as a separable piece of software that performs a generalized data copy operation according to parameters set up and pre-validated by the File Package proper. The separation of function is not complete, particularly in the area of Network interface. Nevertheless, it serves the purpose of breaking the rather massive File Package down into two more easily described parts.

Part IV: UCLA Reccommendations on Libraries in NSW

This section presents UCLA's observations and reccommendations on a fundamental problem that must be solved before NSW can adequately support the use of IBM-compatible software tools. The NSW file system, and thus FP/360, supports only sequential files, while most IBM program-development tools make generous use of "partitioned", or library, files. So under present specifications, FP/360 is not capable of supporting IBM tools in NSW.

PART II

FP/360 -- THE NSW MVT FILE PACKAGE

This section is separately available
as UCLA document UCNSW-204

ILLUSTRATIONS

2. PART II: FP/360

2.1. FP/360 FUNCTIONAL SPECIFICATIONS

Within the National Software Works (NSW), each Tool-bearing host (TBH) is required to have a software component called a "File Package" or "FP", for moving and converting files. This document describes FP/360, a File Package implementation for the IBM 360. Specifically, FP/360 was developed to operate on the UCLA IBM System/360 model 91KK under the MVT Operating System with the Time-Sharing Option, TSO (we commonly refer to this system as OS/MVT). However, with the replacement of certain installation - dependent modules, it will operate on any upward - compatible system.

The reader is assumed to be familiar with reference 1, which prescribes the operation and protocols of an NSW File Package, and with the software environment provided by the NSW.

FP/360 communicates with other NSW processes via the NSW Network Transaction Protocol, or NTP (reference 2, appendix 3), On an IBM system, NTP is implemented on three levels:

* The procedure-call level is implemented by the PL/PCP subroutine package (reference 3).

* The MSG message and direct-connection level is implemented by the PL/MSG subroutine package (reference 4), which also uses the PLOXI package (reference 7).

* The NSWB8 data encodement level is handled by the PL/B8 subroutine package (reference 5).

This document describes in particular Version 2 of the FP/360 implementation.

2.1.1. OVERVIEW

FP/360 functions as an NSW core-system process with generic name "FLPKG". It is essentially a file-copying machine responding to a well-defined set of procedure calls in a well-defined way. Each operation which can be performed by FP/360 is invoked by a single NTP transaction of the form: Generic-request/Specific-reply. Such a transaction, a "procedure call" on an FP procedure, can be expressed in the form:

procedurename (argument-list) -> (result-list)

In particular, FP/360 can execute a procedure call from these remote processes:

2.1.1.1. From a remote process of class "WM" (Works Manager) or "WMO" (Works Manager Operator) FP/360 can execute a call to one of these procedure names:

* FP-EXP (the "Export" procedure call)

* FP-IMP (the "Import" procedure call)

* FP-TRANS (the "Transport" procedure call)

These three procedures, collectively called the "GET procedures", are all concerned with producing a local disk data set filled with the data records of a given file. The source may be either a local data set or a remote FP, i.e., an FP on another host. To retrieve a file across the ARPANET, the local FP contacts an FP instance on the donor host by issuing a subsidiary (FP-SENDME) procedure call for the latter. The two FP's then open a binary simplex connection to pass the data.

The source data may be encoded in IL (Intermediate Language, see Appendix D) or it may be in "clear text", i.e., in one of the many standard disk formats supported by the local operating system, OS/MVT. The output data set is to have specified well-defined local file attributes, but the input data may or may not carry these or compatible attributes. The GET procedures are the most complex parts of FP/360, as the full range of data conversions may be needed.

2.1.1.2. From a remote process of class "CHKPTR" (Checkpointer), "WM", or "WMO", FP/360 can execute a call to the procedure name:

* FP-DEL (the "Delete" procedure call)

This procedure deletes a physical copy from the local disk space (which implies removing its name from the local file directory mechanism).

2.1.1.3. From a remote process of class "FLPKG" (another File Package) FP/360 can execute a call to the procedure name:

* FP-SENDME (the "SendMe" procedure call)

This procedure copies a file from a local disk data set to a remote FP through a binary direct connection. The data will be encoded into IL, and all local file-type dependencies will be stripped from the data; however, no real data conversion is required.

2.1.1.4. The following call is defined, but is not presently used by any NSW process; it is a no-operation in FP/360:

* FP-ANAL (the "Analyze" procedure call)

Later sections will describe each of these operations, with their parameters and results. If any argument list contains more arguments than are known to the selected executor, the excess is discarded without comment. If extensions are defined in an upward-compatible way, this feature will prove useful.

When it is started, FP/360 materializes as an MSG process and issues a ReceiveGeneric for a generic class determined by an initialization procedure (normally "FLPKG"). When the receive completes, FP/360 processes the request for its caller. For a GET call, the local FP may in turn issue a SendMe call to a remote FP. While processing a call, FP/360 is not enabled for new generic calls, and will reject any specifically addressed messages from any process other than its caller or its current callee, by sending an NTP reply with null results and a standard rejection error descriptor.

When it has completed processing, FP/360 returns an NTP reply message to the original caller, and then rematerializes as a new process instance, thus becoming once again receptive to generic calls. It continues to recycle in this manner until a fixed count of cycles, included in the initialization parameters, is exceeded.

2.1.2. PARAMETRIC DATA STRUCTURES

Every call on an FP procedure includes a set of parameters encoded in an NSWB8 LIST (reference 5). While the parameter structure of the call is peculiar to its procedure, many of the elements of that structure are commonly defined. This section gives FP/360's interpretation of these common elements.

2.1.2.1. PCD -- PHYSICAL COPY DESCRIPTOR

For each physical copy of an NSW file, the Works Manager keeps a "Physical Copy Descriptor" (PCD) in its file catalog. A PCD is used by FP/360 in one of four different ways:

1) The Works Manager passes FP/360 a PCD as the definition of the location and IL-encodement of an existing physical copy of an NSW file, or an existing data set outside NSW file space.

2) The Works Manager passes FP/360 a partially filled PCD to identify the directory and/or name under which a new data set is to be created; this is called a "skeleton PCD".

3) FP/360 passes the Works Manager a PCD to define the location and IL-encodement of a newly created data set.

4) FP/360 passes the Works Manager a null PCD as notice that a data set was not created (a null PCD consists of a NSWB8 LIST of count 0).

FP/interprets the fields of the PCD as follows:

* HOST

The HOST field is an NSW host number. This field is only of interest when the PCD is being used to locate an existing data set. FP/360 uses a PL/MSG function (reference 3) to classify the host as LOCAL, FAMILY, or FOREIGN. If HOST is not LOCAL, then only two interpretations of PCD data are possible: 1) the host number can be used in a SendMe call to another File Package; and 2) the ILFLAG field (see below) can be examined.

* DIRECTORY

The DIRECTORY field for a local data set is always chosen by some process on the local host, and it is generally an uninterpretable character string for remote FP's. When the WM calls FP/360 to make a tool copy of a file, the call includes the tool-workspace directory chosen by the local Foreman. In other cases of making a local copy, the local FP should be allowed to choose the directory, so the PCD DIRECTORY field should be null.

For a local 360 file, DIRECTORY contains that part of the local data set name that corresponds to an MVT/TSO LOGON directory. The interpretation of a TSO LOGON directory may vary from one 360 installation to another. At UCLA, the DIRECTORY Field will contain a character string of the form "cccccc.uuu", where "cccccc" is a CCN charge number and "uuu" is the TSO "userid".

LOGON directories with the same account number form a "group", and the directory used to run a job can have group-wide access. Version 2 of FP/360 will have no mechanism to access files outside the directory group in which it was started. Fortunately, it is anticipated that all NSW-related file directories will be in the same group.

If a skeleton PCD has a null DIRECTORY field or is completely null, then FP/360 will use one of two default directories specified by the initialization parameters: NSW filespace default, or non-NSW filespace default.

- NAME

The NAME field contains that part of the local file name (called a "data set name" in MVT) that is not contained in DIRECTORY. An MVT data set name (DSNAME) is formed by catenating these two fields with a period between them.

FP/360 will accept "wild" characters (question marks) in a NAME field, and will generate pseudo-random substitutions to create a unique local name. If a PCD which is required to specify the name for creating a new data set contains a null NAME string, or is entirely null, FP/360 will use a default name from its initialization parameters. Again, there are two defaults, one for NSW and one for non-NSW file space. The default names will generally contain wild characters.

- PHYS

The PHYS field is never examined by FP/360 (see the section entitled "NSW FILE ATTRIBUTES", below). In PCD's generated by FP/360, it will be a character string of count 0.

- ILFLAG

The ILFLAG field is a Boolean value which means "this data set is already physically encoded in IL". FP/360 will use this datum when ranking a set of donor file candidates in a GET procedure.

When the PCD is defining a new data set to the Works Manager, this is the only place where IL-encodement is recorded. Consequently, when the PCD is for an existing data set on the LOCAL host, this datum is the only one that can tell FP/360 whether the data is already IL-encoded.

2.1.2.2. PASSWORD

The NSW PASSWORD parameter is treated differently depending on the corresponding data set's location:

* If the data set is not on the local host, then the interpretation of the password is the responsibility of another File Package. If FP/360 issues a SendMe call to that File Package, the password used in that call will be a copy of the one that FP/360 received from the Works Manager.

* If the data set is locally resident, then the password is intended for gaining access to the specified local directory. However, as noted previously, Version 2 does not allow access to directories which would require passwords (i.e., those in a group different from the one in which FP/360 is running), so the password is ignored.

2.1.2.3. GFT -- GLOBAL FILE TYPE

The NSW Global File Type (GFT) is the symbolic name for a particular set of file attributes. It has the form:

'<host family>-<file type>'

When <host family> is '360', the GFT is said to be "native" to an IBM 360 and hence to FP/360. The File Package on each host family must know all the attributes associated with every native GFT; however, it need not (and must not) assume anything about the attributes associated with a non-native GFT.

In particular, FP/360 includes a table containing the Global File Attributes (GFA's), Local File Attributes (LFA's) and default Physical Structure Attributes (PSA's) for every native GFT. If a native GFT passed to FP/360 does not appear in this table, then a system inconsistency exists, and appropriate local error logging will occur.

2.1.2.4. GFD -- GLOBAL FILE DESCRIPTOR

The NSW Global File Descriptor (GFD) is used by FP/360 to define the Global File Attributes (GFA's) of an NSW file with a non-native file type. Thus, when FP/360 receives a GFT from the Works Manager, there are two cases:

- * If the GFT does not begin with the characters "360-" then it is a non-native type. The accompanying GFD explicitly lists the GFA's of the file. The LFA's are unknown and irrelevant in this case.

- * If the GFT does begin with the characters "360-" then it is a native GFD and the accompanying GFD can be ignored; the GFA's LFA's, and default PSA's for that GFT are taken from the local table.

The contents of the GFD are covered in a later section entitled NSW FILE ATTRIBUTES.

2.1.3. PROCEDURE CALLS SUPPORTED

2.1.3.1. TRANSPORT

The Transport procedure copies a local or remote file into a local disk data set, converting the data to specified target attributes. In the NSW context, the Transport procedure should be used only with source and target files outside the NSW filespace; however, FP/360 can make no check on this.

The form of the Transport procedure call is:

```
   FP-TRANS  (input PCD,
              input PASSWORD,
              input GFT,
              input GFD,

              output PCD,
              output PASSWORD,
              output GFT,
              output GFD)

         ->   ()
```

This call creates a local copy of a local or remote file, using a name determined from the DIRECTORY and NAME fields of "output PCD". Since the actual name used is not returned to the caller, "output PCD" should include a fully specified name. If this name is a duplicate, FP/360 will delete the old copy.

If "input PCD" specifies a remote host, Transport will issue a SendMe call to the FP on that host to retrieve the file.

The operation may fail if it is not possible to translate from the attributes implied by "input GFT" and "input GFD" to those of "output GFT".

Transport returns no reply except the usual completion mode (REPLY vs. ERROR).

Version 2 restriction: the target file, and the source file if local, must be in the NSW directory group.

2.1.3.2. IMPORT

The Import procedure makes an exact copy of a local or remote file into a local disk data set in the NSW filespace directory. No data type conversion is performed; the output file is assumed to have the same GFT (and GFD) as the input file. The form of the Import procedure call is:

```
FP-IMP    (input PCD,
           input Password
           input GFT,
           input GFD,
           output file identifier,
           delete switch)

     ->   (output PCD)
```

If "input PCD" specifies a remote host, Import will issue a SendMe call to the FP on that host to retrieve the file. The output will be encoded in IL if the input is a local file in IL or if it is received from a remote host in IL (Note: It is presently planned to use IL for cross-network transfer of all files, even within the 360/370 family).

If the copy is completed successfully, FP/360 returns an "output PCD" which describes the new data set. In particular, Import takes the DIRECTORY from, and generates a random NAME from, the NSW-filespace default fields of the initialization parameters.

The "output file identifier" is always ignored.

The Boolean parameter "delete switch", if true, specifies that the input file is to be deleted after a successful copy. This option may be set only for a local input file, in which case FP/360 will attempt to implement the procedure as a data set rename. No data movement will occur, and "output PCD" will be a copy of "input PCD" with new values for the directory and name.

2.1.3.3. EXPORT

The Export procedure, like Transport, makes a local copy of a local or remote file, with data type conversion. However, the Export procedure has three additional options, discussed below.

```
FP-EXP    (LIST (input PCD candidates),
           input GFT,
           input GFD,
           output PCD skeleton,
           output PASSWORD,
           LIST (output GFT candidates),
           write secondary output switch,
           output FILE IDENTIFIER)

     ->   (output PCD,
           secondary output PCD,
           output GFT)
```

The "output PCD skeleton" will usually contain a non-null DIRECTORY. The NAME field may contain either a fully specified name or wild characters to be replaced in such a way as to create a unique name. If a fully specified name matches an existing data set, then FP/360 will delete the existing copy. If no NAME is specified, the NAME default for non-NSW filespace will be used. Similarly, if DIRECTORY is not specified, a default non-NSW directory will be taken from the initialization parameters.

The three additional options of Export are:

1) Export chooses the input file from "input PCD candidates". It will order this list of input candidates by estimated ease of copy, using this simple preference definition:

    1) a local data set not encoded in IL.
    2) a local data set encoded in IL.
    3) a remote data set encoded in IL.
    4) a remote data set not encoded in IL.

   Having formed this sorted list of input PCD's, FP/360 loops down the list and attempts to copy each in turn, until either: 1) a successful copy is produced; 2) the list is exhausted; or 3) the number of attempts exceeds a limiting value acquired as an FP/360 initialization parameter. Setting that parameter to 1 effectively disables retry.

2) From "output GFT candidates", Export must choose a single GFT for its primary output.

* If the list of output GFT candidates is empty, then the input GFT will be used.

* If the input GFT appears in the list of output GFT candidates, it will always be selected.

* Otherwise, the list of candidates is sorted into order of increasing cost of conversion, while preserving the original order in cases of equal cost (the list was originally ordered by the caller's preference). The algorithm for this sort is described in the section entitled "CONVERSIONS IN FP/360". The first GFT on the sorted list is selected.

* It is possible that none of the conversions from the input GFT to any of the output GFT's are possible. In this case, the entire Export operation is failed.

3) Export can create a secondary output file in the same format and with the same type as the input.

If "write secondary output switch" is true, FP/360 is requested to create a secondary copy. However, Export has the privilege of refusing to do so if the copy would be redundant, due to the existence of a local data set among the input PCD candidates. Refusal is indicated by returning a null "secondary output PCD" to the Works Manager.

Otherwise, Export will create a data set containing the records exactly as they are received from the donor file package. The name for the secondary output data set is always generated by FP/360 using the same mechanism described earlier for naming the result of the Import procedure, but using the NSW filespace defaults in the initialization parameters.

Version 2 restriction: the entire Export procedure will fail if any unrecoverable error occurs, even one not preventing producing the primary output data set.

2.1.3.4. SENDME

The SendMe procedure copies a file from a local disk data set to a remote FP through a binary direct connection. The data will be be encoded into NSW Intermediate Language (IL), and all local file-type dependencies will be stripped from it. However, no data type conversion is performed -- the output GFT is identical to the input GFT.

The form of the SendMe procedure call is:

```
FP-SENDME (input PCD,
           input PASSWORD,
           input GFT,
           input GFD,
           receiver host number,
           maximum byte size,
           maximum block size,
           family argument)

      ->  (connection identifier,
           actual byte size,
           actual block size,
           file size,
           family reply)
```

The "input GFD" is actually redundant. Either the data to be transmitted is of a native type, in which case its attributes are known, or it is in IL, in which case no attributed will need to be known to transmit it. So this datum is effectively ignored.

The actual block size will be the minimum of: 1) the requested maximum block size; and 2) a limiting value acquired as an FP/360 initialization parameter. At present, transmission block sizes are established by Gentlemen's agreement, and will not vary. Therefore, if the input is IL-encoded, and if one of its pre-formatted IL transmission blocks exceeds this block size, the procedure will be aborted.

The "file size" result will be the bit size of the actual disk allocation on the local disk, adjusted, if the data set is not already in IL, by its LFD's "compression factor" attribute.

Version 2 restrictions:

* The receiver host number is already known, so the corresponding parameter is ignored.

- "Connection identifier" is always 1.

- "Actual byte size" is always 8.

- Non-IL transmission is not supported: therefore, "family argument" is ignored, and "family reply" will always be EMPTY.

- SendMe cannot generate alarms. Any terminal error condition will be signalled by closing the direct connection without sending the end-of-transmission indicators.

2.1.3.5. DELETE

The Delete procedure call is the only "collective" operation implemented in FP/360. The form of the Delete procedure call is:

```
FP-DEL (LIST (local pcd))

     -> (LIST (error descriptor))
```

where the arguments are physical copy descriptors defining the data sets to be deleted, and the result-list is either empty or a list of corresponding error descriptors. The possible results are:

* If all specified deletions are successful, the entire transaction completes in REPLY mode (reference 3), and the result-list is replaced by a LIST of count 0.

* If there is an error that relates to the procedure call as a whole, the transaction completes in ERROR mode (reference 3) and the result-list is a LIST of count 0.

* Otherwise -- if there is one or more errors relating to the deletion of specific data sets in the argument list -- then the entire transaction completes in ERROR mode (reference 3), with the main NTP error descriptor specifying "partial results returned". In addition, the result-list contains a result descriptor for each specific PCD. Each of these descriptors is either: a null list, if the deletion was successful, or a list of the form

  ```
  LIST (errorclass, errnumber, errorstring)
  ```

  Notice that errors associated with a single PCD have no effect on the processing of other PCD's.

Version 2 restriction: The Checkpointer is now sending the Delete call using another syntax -- the single PCD is not enclosed in a list. For now, that form is the one recognized by FP/360.

2.1.3.6. ANALYZE

The Analyze procedure is currently incompletely defined. Therefore, in FP/360, Analyze is a no-operation corresponding to the form:

```
FP-ANAL () -> ()
```

2.1.4. NSW FILE ATTRIBUTES

An NSW file is really an abstraction, standing for a collection of equivalent physical copies. The location of each physical copy is defined by an NSW data structure called a Physical Copy Descriptor, or PCD. All physical copies of the same NSW file share the same Global File Attributes, or GFA's.

The GFA's of an NSW file, and thus of the data in a local copy, are passed to an FP in the form of a character string called a "Global File Type", or GFT. This string consists of a prefix part which is the NSW "host family name" ("360" for the family to which FP/360 is native), followed by a hyphen, followed by a suffix part chosen to be unique and mnemonic within the "family". Such a name represents very nearly the complete set of data attributes that a particular FP must know about the local copy.

In FP/360, attributes are structured into three discrete levels; however, it should be recognized that the assignment of attributes to one level or the other is more an engineering (if not political) decision than a theoretical consequence. As a result of future experience with the NSW, additional attributes may be added to the global set, the driving force being tool installers and users who want data type mismatches to be handled automatically by the NSW mechanism.

* Global File Attributes

Global File Attributes (GFA's) are basic ones that apply to the data within a file, whether it is represented in IL or not. These must be the same for all copies of that file. They are uniformly defined across all NSW host families. The character/binary distinction is a good example.

While these attributes are strictly implied by the GFT, their derivation is always performed by the Works Manager, in order that FP/360 need not be aware of the meanings of GFT's not native to the 360 family. These derived attributes are packaged into the Global File Descriptor, or GFD. A GFD is always shipped along with a GFT when the Works Manager sends the GFT to FP/360, with one exception: the output file type of the Export procedure is represented only by a GFT because that GFT is guaranteed to be native to the 360 family. FP/360 keeps a table of the attributes of all native types, and this table includes the information in the corresponding GFD's.

* Local File Attributes

Local File Attributes (LFA's) describe the way that data of a given type is represented in non-IL ("clear text") form within the 360 family. The columnar position of a key field is a good example. These attributes are derived from the GFT for any native type by FP/360. The LFA's provide the instructions needed by FP/360 to translate data between IL string encodement and the clear text encodement implied by the GFT.

* Physical Structure Attributes

Physical structure Attributes (PSA's) describe the specific mapping of a data set on disk. On an IBM 360, the DCB parameters are a good example.

PSA's are handled differently depending on whether FP/360 is assigning them to a newly created data set or determining those already assigned to an existing data set. In the former case, default values can be derived from the GFT and embellished by anything known about the quantity of data the file is expected to contain. In the latter case, most PSA's are stored by the TBH operating system as part of the data set label, and are available to FP/360 on request.

NSW provides that PSA's that are not automatically available will be kept by the Works Manager in a PCD field named PHYS. The PCD is always available whenever FP/360 accesses an existing file, so the PHYS field information is always available when it is needed. However, since the PSA's used by FP/360 are kept by the local operating system, FP/360 currently has no need for the PHYS field.

One special case should be noted. For each physical copy, an FP will need to know whether it is physically encoded in IL. Since this attribute is not kept by any existing host system, and since it is meaningful across all NSW host families, it is kept in a special field of the PCD named ILFLAG.

2.1.4.1. GLOBAL FILE ATTRIBUTES

As noted earlier, FP/360 may obtain the GFA's from the GFD (for a non-native type) or from its own local table (for a native type). In either case, FP/360 interprets the GFA's in the following manner:

2.1.4.1.1. CLASS

This field determines whether the data consists of characters or binary bytes, with the following consequences.

* Character-class data represents an array of ASCII graphics of dimensionality between 1 and 4 (see the dicussion of dimensionality below). A full complement of format effectors is defined for use in positioning graphics within the array. Unspecified array positions are assumed to contain the fill character "blank", which is also used for optimal compression in IL.

  Data with dimensionality of 2 or higher is organized into "records", with which there may be associated character-string "keys". A common use of these keys is to record the "sequence numbers" associated with text lines by some text editors, compilers, etc.

* Binary-class data is of dimensionality 1 or 2, representing either a single byte string or a sequence of (short) byte strings called "records", respectively. There are no format effectors other than the record separators. In two-dimensional data, a record may be associated with a character-string key as well as the binary text.

  For binary-class data, the "fill" character used for IL compression is a byte of binary zeros.

2.1.4.1.2. KEY DEFINITIONS

When keys are associated with data records of the file, the keys are always character strings. The GFA's for keys are a Boolean "keys present" indicator and an integer "key length" field.

2.1.4.1.3. VARIABLE FORMAT EFFECTORS

Format effectors can be classified as regular and irregular, with the regular ones further classified as horizontal or vertical, as interval or absolute, as positive or negative, and as fixed or variable (see figure 1). The fixed format effectors are interpreted the same for all files and by all FP implementations (by system-wide convention, Carriage Return and Backspace are always considered to be non-destructive). The variable format effectors are defined in the GFD under the name TAB-DESCRIPTOR and can thus be interpreted the same, for a given file type, by all FP implementations.

FP/360 will be able to support expansion of all defined format effectors received from remote FP's. However, when encoding files of native global types, FP/360 will generate only regular positive fixed forms and the irregular "SKIP(0)" form, i.e., only IL types.

Figure 1: Classification of Format Effectors

REGULAR HORIZONTAL TYPES:

Interval positive (d>0)--	(Variable):	HT as interval
Interval negative (d>0)--	(Fixed):	Destructive BS (not in ASCII)
Absolute positive (d>1)--	(Variable):	HT as stop list
Absolute negative (d>1)--	(Fixed):	Destructive CR (not in ASCII)

REGULAR VERTICAL TYPES:

Interval positive (d>1)--	(Variable):
	(Fixed):
		LF, VT as interval, or
		IL "skip n" (n>0)
		record control.

Interval negative (d>1)--	(fixed):	Inverted linefeed (not in ASCII)

Absolute positive (d>2)--	(Variable):
	(Fixed):
		FF, VT as stop list, or
		IL NewPage

Absolute negative (d>2)--	(none defined)	

IRREGULAR TYPES:

(d>3)--	Non-destructive backspace
(d>3)--	Non-destructive carriage return
(d>3)--	IL "skip 0" record control

2.1.4.1.4. DIMENSIONALITY

FP/360 copies a file in the form of a one-dimensional stream of characters or binary bytes. However, this stream is understood to represent an array of more complex structure, with up to four meaningful dimensions. The logical equivalence of physical copies of the same NSW file is properly stated in terms of equivalence of the multidimensional array rather than that of the stream used for transmission. Thus FP/360 must concern itself with preserving the integrity of that array. In particular, FP/360 interprets the dimensionality in the following manner:

2.1.4.1.4.1. DIMENSIONALITY = 1

One-dimensional data consists of a stream of bytes (or characters) that are not logically grouped into lines or records. The single dimension corresponds to file size, and is effectively unbounded.

```
{ BYTE [ c ],    c= 1 to file_size }
```

For character-class files, regular horizontal format effectors (see Figure 1) are possible, but no other format effectors would be meaningful. The data may be broken arbitrarily into record-like strings for convenience in handling, but it is understood that these strings are not logical records.

IL "record control" fields have no meaning; FP/360 will ignore them when receiving and will generate "SKIP(0)" when transmitting.

Logical equivalence of one-dimensional file copies is defined to be equivalence of the byte or character streams represented by the encodement (i.e., after IL expansion), regardless of the class of the data.

A one-dimensional file cannot have keys.

2.1.4.1.4.2. DIMENSIONALITY = 2

Two-dimensional data consists of a stream of bytes (or characters) divided into records or lines. Keys are permitted, and if they appear there is a key included with each record. The first dimension is bounded by the "Record Length Range" datum of the LFD, but the second corresponds to file size, and is effectively unbounded.

```
{  KEY [ k, r ], BYTE [ c, r ],

     for:   c= 1 to max-record-text-length,
            k= 1 to key-width,
            r= 1 to record-count,          }
```

For character-class files, it is possible to define any kind of regular horizontal format effectors and regular vertical interval format effectors, but no other kind are meaningful.

Equivalence of two-dimensional file copies is defined to be equivalence of the two right-ragged arrays represented by the data encodement (i.e., after IL expansion), and when appropriate, of corresponding keys. The right edges are defined to include trailing "fill characters" as a part of the data.

2.1.4.1.4.3. DIMENSIONALITY = 3

Three-dimensional data consists of a stream of characters, grouped into lines, which are then grouped into pages. Keys are legal, but will probably be rare. The first dimension is bounded by the "Record Length Range" datum of the LFD, and the second by the "page depth" datum, but the third corresponds to file size, and is effectively unbounded.

```
{  KEY [ k, r, p ], BYTE [ c, r, p ],

     for:   c= 1 to max-record-text-length,
            k= 1 to key-width,
            r= 1 to page-depth,
            p= 1 to page-count             }
```

Only character-class data can be three-dimensional, and it is meaningful to define all regular format effectors.

Equivalence of three-dimensional file copies is defined to be graphical equivalence of the two arrays represented by the data encodement (i.e., after IL expansion), and when appropriate, of corresponding keys. The right edges are ragged in all dimensions, and are defined to exclude trailing "fill characters" from significance as data.

2.1.4.1.4.4. DIMENSIONALITY = 4

Four-dimensional data consists of a stream of characters, grouped into records, which are then grouped into lines, which may then be grouped into pages. Keys are legal, but will probably be rare. The first dimension is bounded by the "Record Length Range" datum of the LFD, and the second corresponds to overprinting and is unbounded. The third dimension is bounded by "page depth", while the fourth corresponds to file size and is also unbounded.

```
 { KEY [ k, r, l, p ], BYTE [ c, r, l, p ],

     for:   c= 1 to max-record-text-length,
            k= 1 to key-width,
            r= 1 to max-overprint-depth,
            l= 1 to page-depth,
            p= 1 to page-count          }
```

Only text-class data can be four-dimensional, and it is meaningful to define all regular and irregular format effectors.

Equivalence of four-dimensional file copies is defined to be graphical equivalence of the two arrays represented by the data encodement (i.e., after IL expansion), and when appropriate, of corresponding keys. The right edges are ragged in all dimensions, and are defined to exclude trailing "fill characters" from significance as data.

2.1.4.1.5. BYTESIZE

A file may consist of bytes of a width in the range 8 - 255 bits; however, FP/360 will refuse to process files with a bytesize other than 8.

2.1.4.2. LOCAL FILE ATTRIBUTES

FP/360 gets its LFA's from a Local File Descriptor (LFD) which is stored locally and retrieved via the GFT. The fields of this descriptor are listed below.

It is important to understand that values for certain LFA's are often required by FP/360 even when thay do not appear to have meaning for the particular data type. This is because they may be needed to perform a type conversion into that type from a non-native type about which nothing is known. For example, a "page depth" datum is tabulated for a two-dimensional data type if conversion of non-native three-dimensional data into that type could occur, even though page depth has no meaning for an existing two-dimensional file.

In the following, some variables have as a value an indicator that user permission is to be obtained via the NSW HELP mechanism. In Version 2, that mechanism is not available to FP/360, so these permissions are assumed to be granted.

Certain of the LFA's are now required by NSW convention to be set in certain ways. For instance, it has now been decided that trailing fill characters are <u>always</u> significant in one- or two-dimensional data.

2.1.4.2.1. Dimensionality preference -- this datum defines a preference ordering of the four dimensionalities for situations where dimensional conversion may be required. For each, a flag indicates whether conversion from that dimension is permitted, forbidden, or permitted only with explicit user permission.

2.1.4.2.2. Min Record Length, Max Record Length -- These two fields give the minimum and maximum number of bytes of text (exclusive of keys) that a record of this type can contain.

2.1.4.2.3. Short record handling -- pad, signal error, or ask user.

2.1.4.2.4. Long record handling -- truncate, fold (make two records), ask user, or signal error.

2.1.4.2.5. Record fold margin -- what column continuations begin in.

2.1.4.2.6. Page depth -- the number of lines (not records) to a printer page.

2.1.4.2.7. Short page handling -- pad, leave as-is, or ask user.

2.1.4.2.8. Long page handling -- truncate, fold (make two pages), leave as-is, or ask user.

2.1.4.2.9. Key position -- 1-origin index of the text-field byte before which the key is to appear.

2.1.4.2.10. How to generate missing keys -- don't (signal error), count records, blank fill, delete field, or ask user.

2.1.4.2.11. Option switches:

Force upper case
Suppress code translation
Suppress IL expansion
Input-only type
Trailing fill characters are significant

2.1.4.2.12. Format effector handling switches:

Horizontal tab handling:
Leave as tab code,
Expand by input GFD, or
Expand by output GFD.

Vertical format effector handling (For each of VT, LF, and FF):
Leave as EBCDIC code,
Expand by input GFD, or
Expand by output GFD.

Backspace handling:
Leave as backspace code,
Expand destructively, or
Expand non-destructively.

Carriage return handling:
Leave as carriage return code,
Expand destructively, or
Expand non-destructively.

2.1.4.2.13. Compression factor -- typical IL bytes/"clear text" bytes.

2.1.4.3. PHYSICAL STORAGE ATTRIBUTES

FP/360 gets its PSA's for an existing data set from the data set label. When creating a new data set, recommended PSA's are tabulated in a descriptor (the PSD) which can be retrieved via the GFT. Its fields are:

* DSORG -- the data set organization.

  FP/360 will support only Physical Sequential (DSORG=PS) in Version 2.

* RECFM -- the record format.

  In Version 2, FP/360 will support:

  ```
  F [B[S]][A] (fixed-length records)
  V [B][S][A] (variable-length records)
  U [A]       (undefined-length records)
  ```

* OPTCD -- data managment option codes.

  FP/360 will not support any of these in Version 2.

* LRECL -- logical record length.

* BLKSIZE -- physical block size.

  Two values are tabulated. FP/360 may choose a value within that range which is compatible with RECFM and LRECL, and which optimizes utilization of the selected physical device. If the two values are the same, then no variation in block size is allowed.

* KEYLEN -- length of random-access retrieval key.

* RKP -- offset of random-access retrieval key.

  Version 2 does not support random access to data sets, so non-zero values of KEYLEN and RKP will not occur.

* SPACE -- recommended allocation if no size data is available.

  There are three fields -- PRIMARY, SECONDARY, and DIRECTORY. The first and second fields are inital and subsequent allocation quantities in selected-blocksize units. The third field is relevant to partitioned data set organization, and so is not currently supported. It will always be zero.

2.1.5. MAPPING FILES ON A 360

Under FP/360, a file encoded in IL is recorded on disk according to these conventions:

1) The data set organization is Physical Sequential (DSORG=PS).

2) The record format is Variable Blocked (RECFM=VB) or Variable Blocked Spanned (RECFM=VBS), depending on the values selected for LRECL and BLKSIZE.

3) Each logical record is one IL transmission block. The maximum logical record length (LRECL) is four greater than the corresponding SendMe procedure transmission block size, since disk records have count and control fields four bytes long, and the transmission block size does not include even the 2-byte IL count fields.

4) The data set maximum block size (BLKSIZE) is independent of the data. It is selected by FP/360 by choosing a device-optimizing value between two limits provided it as initialization parameters.

5) The data in the data set can be considered free of any LFA's, <u>whether</u> <u>or</u> <u>not</u> its GFT is 360-native.

2.1.6. CONVERSIONS IN FP/360

A Transport or Export operation may create a new file with a different GFT than the input file; the change of the file contents as a result is called "conversion" of the file. Notice that such type conversion may take place only when the target file is outside NSW file space
°the original GFT is always preserved within NSW file space.

2.1.6.1. TRANSLATABILITY

The translatability of a file is a function of its existing GFT and the desired new GFT, or, more precisely, of an input GFD and an output GFT, GFD, and LFD. A legal translation will have all the following properties, and need have none other:

1) The output GFT is native to the 360 family.

2) The input and output data classes (binary vs. character) are the same.

3) The input dimensionality is one that is permitted by the output LFD.

4) Either the input has keys, or the output does not have keys, or the output's "how to generate missing keys" datum doesn't contain the value "don't".

5) The Bytesize fields match.

For purposes of selecting the primary output GFT which is "best", FP/360 defines a "dimensionality preference" table in the GFD for each output type. The table contains four entries, one for each of the possible input dimensionalities. An entry actually consists of two parts: a three-state translatability flag with values:

* "permitted"

* "permitted with user permission"

* "not permitted"

and a preference-rank number (ignored for "not permitted" entries). Version 2 of FP/360 will not support asking users for permission, so those in the second category are treated as "permitted", but with lower preference than those in the first category. Version 2 will therefore select the target GFT with "permitted" dimensionality and the highest rank number, or if none are "permitted", the one "permitted with user permission" and highest rank number.

2.1.6.2. DIMENSIONAL CONVERSION

When a file is entered into NSW file space, its GFT and therefore its dimensionality attribute must be declared. This attribute specifies the maximum dimensionality that the file might have, and that the eventual user of the file must be prepared to handle; however, the actual file contents might in fact be of lower dimensionality. If a host "lies" and declares all its files to be of dimensionality 4, many tools may refuse to process these files as input. However, there may be no harm in a paper-tape host declaring every file to have dimensionality 3.

Any data of dimensionality "n" is also legal data of dimensionality "n+1", where the bound of the new dimension is one. However, the converse is not true. In particular, when FP/360 is instructed to reduce the dimensionality of a 4-dimensional file, it must make potentially destructive changes. By external conventions, all native types permit conversion out of any higher dimensionality only with explicit user permission; however, if there is no user to ask, the conversion is permitted.

In general, a conversion which lowers the dimensionality by n can be defined in terms of a series of n conversions, each lowering the dimension by 1. In all cases the surfaces of the unwanted dimension collapse *into a plane*, somewhat as does a closing Venetian blind. This has the correct default property: if the data is in fact of the lower dimensionality already, it will be unchanged by the transformation.

2.1.6.2.1. CONVERTING 4 TO 3

This is an information-destroying conversion. Each page consists of a primary page surface and an unbounded number of overprint page surfaces. Cross sections of these surfaces are lines. The intersection of a line and a page surface is a record. In most cases, the overprint dimension is very narrow and very ragged.

For every line, each non-null overprint record is meshed into the primary page surface by pushing down all subsequent lines, including null ones. The page is thus reduced to a single surface. If the depth of this surface exceeds the upper bound of the page-depth dimension, the "long page handling" datum of the output LFD is queried. If this has the value "truncate", excess lines are discarded. If it has the value "fold", a new page surface is constructed of the excess lines, and this surface is placed behind the current page by pushing all subsequent pages back one. If it has the value "leave as-is", the page-depth bound is effectively (but temporarily) increased to accomodate the long page.

The above procedure is implemented by replacing "skip 0" record control with "skip 1". If page overflow occurs, either records are discarded until the next "formfeed" record control, a "formfeed" is forced into an existing record, or the situation is simply ignored.

2.1.6.2.2. CONVERTING 3 TO 2

Each page consists only of a single surface. These surfaces have a fixed maximum line count, but vary in actual line counts. Each page is catenated to the bottom of the previous page. If the page being catenated to has fewer than the maximum lines, and if the output LFD "short page handling" datum has the value "pad", then null lines will be inserted to bring the short page up to size.

In other words, "formfeed" record control is converted to "skip n", where "n" is either 1, or 1 plus the output page depth minus the line counter.

2.1.6.2.3. CONVERTING 2 TO 1

Each record is catenated after the previous one, resulting in a single string. Keys are discarded. IL record control fields of the form "skip n" are replaced by a string of length n-1 times the output's "minimum text length" field, and containing the selected fill character. In practice, it is then necessary to break the resulting string into arbitrary transmission block strings, and prefix these with meaningless "skip 0" fields.

FP/360 will not actually support this conversion in Version 2.

2.1.6.3. CASE CONVERSION

If the "force upper case" option switch is set for the output file type, and if type translation is in effect (that is, if the input and output file types are not equal), then case conversion occurs. The EBCDIC codes for the lower-case characters ("a" - "z") are converted to those for the upper-case characters ("A" - "Z"). No other character codes are affected.

2.1.6.4. TRUNCATION AND PADDING

When type translation is in effect (that is, if the input and output file types are not equal), then record and page truncation and padding may occur. These conversions are governed by the indicators set in the local file attributes. They are implemented as described above under "DIMENSIONAL CONVERSION".

2.1.6.5. FORMAT EFFECTOR EXPANSION

When translating a file out of IL representation, and in no other case, format-effector conversion occurs. This conversion is defined to operate on those ASCII format effectors that the encoding File Package has seen fit to represent in the stream by tokens of the corresponding explicit IL format-effector type codes. The data characters of an IL stream are fully transparent, so format effectors that have been left in that representation are simply translated into their EBCDIC equivalents.

The ASCII format effectors that are converted are: form feed (FF), line feed (LF), vertical tab (VT), horizontal tab (HT), backspace (BS), and carriage return (CR). Each occurrence of these codes is converted without regard to any pairing. It is a requirement on the encoding File Package that pairs of format effectors that are equivalent in meaning to the IL "new line" ("skip") and "new page" record control constructs be represented by those constructs.

Each format effector has a corresponding local attribute that defines how it is to be expanded. The details of the expansion are left for a future version of this document.

2.2. FP/360 PROGRAM LOGIC

FP/360 has the overall structure shown in figure 2. The dispatcher establishes process instances, accepts procedure calls, and selects a procedure executor. Each procedure executor is responsible for decoding that procedure's parameters, and usually, for completing the main transaction by encoding appropriate results. The first action of an executor is thus to invoke the PL/B8 package (reference 5) to decode the NSWB8 string contaning the parameter list, according to the particular syntax of that procedure call.

Ignoring for the moment the trivial functions, we can describe FP/360 as primarily a software machine for executing a copy operation on a data file, possibly producing two outputs for a single input. Following this model, most of the procedure executors invoke a "back-end" component called the Basic Copy Machine, or BCM (reference 8), after parametrically tailoring it for the particular procedure being executed. The BCM sports a variety of mode switches by which it can be parameterized to perform one of at least three basic copy types: local copy, remote get, or remote send. Similar switches control conversion of data among three possible forms: clear text, IL-encoded, and "normalized", an intermediate encodement internal to the BCM.

Because of the complexity of the BCM, it is separately documented -- see reference 8.

Further discussion of the logic of FP/360 is deferred for a future version of this document.

Figure 2. FP/360 Structure

```
*********************
*                   *
* FP/360 DISPATCHER *
*                   *
*********************
      |
      |        ***************************
      |        *                         *
      *------->*    ANALYZE EXECUTOR     *
      |        *                         *
      |        ***************************
      |
      |        ***************************
      |        *                         *
      *------->*    DELETE EXECUTOR      *
      |        *                         *
      |        ***************************
      |
      |        ***************************
      |        *                         *
      *------->*    SEND EXECUTOR        *--------*
      |        *                         *        |
      |        ***************************        |
      |                                           |
      |        ***************************        |
      |        *                         *        |
      *------->*    TRANSPORT EXECUTOR   *--------*
      |        *                         *        |
      |        ***************************        |
      |                                           |
      |        ***************************        |
      |        *                         *        |
      *------->*    IMPORT EXECUTOR      *--------*
      |        *                         *        |
      |        ***************************        |
      |                                           |
      |        ***************************        |
      |        *                         *        |
      *------->*    EXPORT EXECUTOR      *--------*
               *                         *        |
               ***************************        |
                                                  V
                                    *********************
                                    *                   *
                                    *  BASIC COPY       *
                                    *         MACHINE   *
                                    *                   *
                                    *********************
```

2.3. APPENDIX A: STATUS OF FP/360 IMPLEMENTATION

2.3.1. CURRENT RESTRICTIONS AND DEFERRED FEATURES

The following features of the current FP specification have not been implemented, or have been incorrectly implemented in Version 2 of FP/360. This list is roughly ordered by decreasing importance and/or increasing cost of implementation.

2.3.1.1. FORMAT EFFECTORS

All format effectors and record control tokens of IL are implemented. However, those whose interpretations are defined in the GFD (HT, VT, LF, and FF) are supported only in their interval form. That is the only form ever used by the other host families that FP/360 must support at this time.

2.3.1.2. ALARMS

FP/360 never arms itself for alarms, and it never sends an alarm; however, the status of alarms in the current File Package specification is in flux anyway. In the meantime, FP/360 has no mechanism for reporting the status of a transfer operation. If an error condition is found during data transfer, FP/360 will immediately close the connection, without sending the required in-band normal-eod signal.

2.3.1.3. ERROR DESCRIPTORS

Full error descriptors are not supplied by FP/360 due partly to restrictions in the current version of the PL/PCP package (reference 3), which FP/360 uses for transaction management. In particular:

* The optional parts of an error descriptor are always null.

* Only one error can be reported -- the first one detected.

* The values of the fault class and fault number fields have not been properly correlated with other FP implementations.

* An error descriptor received by FP/360 from an imbedded SendMe transaction is not copied into the reply that completes the main transaction. The reply will indicate only that an error occurred in "SendMe".

2.3.1.4. STREAM FILES

Conversion to or from one-dimensional files is not supported.

2.3.1.5. FAMILY COPIES

A format for family copies of files which cannot be described in IL has not been defined for the IBM 360 family. All network transmission uses IL.

2.3.1.6. PASSWORD PARAMETER

A local data set can be accessed by the FP only if it exists within a directory in the NSW directory group (i.e., having the NSW charge number). Since there is no mechanism to "connect" to a non-NSW directory, the password parameter is ignored for local data sets.

2.3.1.7. IL REBLOCKING

IL reblocking is not supported; a request to send an IL-encoded file with a transmission block size smaller than the IL blocksize in which it is recorded on disk may fail. This is not expected to be a problem, since IL block sizes are not expected to vary in the near future.

2.3.1.8. BYTE SIZE

Only byte size 8 is supported.

2.3.1.9. SUBFILES

The "subfile" facility of IL is not supported.

2.3.1.10. ANALYZE

The FP-ANAL procedure call is a no-operation, and FP/360 itself never issues such calls.

2.3.1.11. PHYSICAL FORMAT RESTRICTIONS

Only sequential (DSORG=PS) files are supported on the 360. In particular, partitioned (library) files are not supported, nor are generation data groups.

The MVT option codes (OPTCD) are not supported.

Direct (non-sequential) access to a keyed data set is not supported.

2.3.1.12. DEVICE TYPES

The filespace must be on permanently-resident direct-access volumes; tapes and removable disk packs are not supported.

2.3.1.13. HONESTY CHECKS

During translation or re-encodement of a file, FP/360 does not verify that the input data conforms to the advertised dimensionality. However, the result created by FP/360 will have the requested dimensionality, regardless of input. There is one major exception: if the input and output files are of the same type and encodement, then no data interpretation occurs during the copy, and no checking of any kind is done.

2.3.2. FP SPECIFICATIONS QUESTIONS

This section lists design features of FP/360 which are at variance with questionable or unsettled aspects of the official FP specs. They are listed here to draw attention to some areas of uncertainty in the specifications.

2.3.2.1. OVERWRITING EXISTING DATA

If FP/360 is asked to overwrite an existing data set, it will delete the existing copy and create a new data set. This approach has been taken because there is now no way to avoid it and make NSW work. However, we believe that there are scenarios where this destruction of existing data may be an accident. One would hope a conscientious WM would be concerned about clobbering data accidentally.

2.3.2.2. EXPORT FAILURES

When FP/360 Export is creating both a primary (exported) copy and also a secondary NSW copy, it fails if either copy fails, even if the other copy could have been created successfully. This could waste an expensive and lengthy network transmission. We suggest that the FP specifications be changed to allow a partial success in Export.

2.3.2.3. RESULT LISTS

In general, when FP/360 encounters an error, it will abandon the operation completely rather than complete it partially. Therefore, the "result list" for the FP calls will be null. Eventualy, it may be desirable to define a restart mechanism to salvage partial file transfers.

An exception is the Delete operation. FP/360 will handle a list of deletions, returning a result list that indicates which ones succeeded and which ones failed.

2.3.2.4. SYNTAX OF DELETE

The syntax of the Delete transaction is implemented according to what is now being received from the Checkpointer, not according to the FP specifications. The Checkpointer is due to be changed in the future.

2.3.2.5. READ-ONLY FILES

NSW specifications state that a tool may have access to only a copy of an NSW file; however, there are a number of potential 360 tools which are incapable of writing to their input files, and which can be reliably expected not to clobber them. For these tools, we need to avoid the delay inherent in making an unnecessary copy. We now do this using the "read-only" local file attribute, but this method is unsatisfactory both in design and implementation. We need an NSW-wide specification for this facility.

2.3.2.6. AVOIDING REDUNDANT LOCAL COPY

If Export finds a local NSW copy of the source file, it does not produce a new NSW copy even if requested to do so. We believe that the FP specifications should state this.

2.3.2.7. ASKING USER ABOUT CONVERSIONS

When a requested file conversion implies a non-invertible change of the logical file contents, we wish to make a HELP call to ask the user's permission. This facility is not presently available to us, so FP/360 assumes that the permission is granted. We believe that the HELP facility should be made available to a File Package whenever there is a User available.

2.3.3. DESIGN POINTS

The following are particular aspects of the FP/360 design that we believe to be permanent and non-controversial. They are listed here just because someone may want to know.

2.3.3.1. FILE NAMING CONVENTIONS

The catenation of the PCD's DIRECTORY and NAME fields, with a period between, must form an MVT DSNAME of at most 44 characters.

2.3.3.2. USAGE OF PCD PHYS FIELD

The PHYS field of the PCD is never examined. In locally created PCD's, it will be a null character string.

2.3.3.3. SELECTING CANDIDATE FILE FOR EXPORT

Export will prefer input PCD candidates in this order:

1) a local data set not IL-encoded.
2) a local IL-encoded data set.
3) a remote IL-encoded file.
3) a remote file not IL-encoded.

2.3.3.4. EXPORT RETRY

If FP/360 Export encounters a failure in retrieving a particular physical copy of a file, it will select the next most desirable copy from the PCD list and try again. This will continue until a copy is produced, the PCD list is exhausted, or the number of retries exceeds a limiting value which is an initialization parameter.

2.3.3.5. UNDEFINED FORMAT EFFECTORS

FP/360 will support variable format effectors only when they are explicitly defined in the GFD. If such a format effector occurs without a GFD definition, it will simply be converted into the corresponding EBCDIC code. However, such codes are included in the EBCDIC set only for physical device control; they are normally unacceptable input to 360 tools.

2.4. APPENDIX B: 360 FAMILY CONVENTIONS

At present, FP/360 does not exist on more than one host; therefore, no actual intra-360 communications techniques are defined. The items below are only directions which might develop into family conventions.

2.4.1. All line transmission, even family copies, will probably use the IL encodement. We believe that a true family transmission protocol will become useful only when special file structures such as IBM's Partitioned Data Set (PDS) organization are supported. Even then, we would propose to use a superset of IL as the family protocol.

2.4.2. The PCD DIRECTORY filed contains that part of the local data set name that corresponds to an MVT/TSO LOGON directory. This is not absolutely constant across MVT implementations, so, for the purpose of defining a 360 family convention, no more will be said.

2.4.3. The NAME field contains that part of the local DSNAME that is not in DIRECTORY. The DSNAME is formed by catenating these two fields with a period between them.

2.4.4. The PCD ILFLAG field is the only place where FP/360 records or discovers the fact that a local data set is in IL.

2.5. APPENDIX C: VERSION 2 INITIALIZATION PARAMETERS

FP/360 decodes a set of initialization parameters from a configuration data set which may optionally be supplied under file name (DDNAME) PARMS. This data set is in the form of a PL/I GET DATA input stream. The following data may be specified, where each name should be qualified by the name "P.":

Name:	Type:	Default:	Meaning:
NSW_DIRECTORY	CHAR	'AHA179.NSW'	Default directory name for creating a new data set in NSW filespace.
NSW_DSN_PAT	CHAR	'GEN.NSW?????'	Default name used for creating a new data set in NSW filespace.
WSP_DIRECTORY	CHAR	'AHA179.NSW'	Default directory name for creating a new dataset outside NSW filespace.
WSP_DSN_PAT	CHAR	'GEN.WSP?????'	Default name used for creating a new data set outside NSW filespace.
GENERICNAME	CHAR	'FLPKG'	FP/360's MSG generic name.
MSG_TIMEOUT	FIXED	60,000	MSG message timeout value, in 0.01 seconds.
PCP_TIMEOUT	FIXED	600,000	PCP transaction timeout value, in 0.01 seconds.
MAXCOPIES	FIXED	1	Limit on the number of copy attempts within Export.
MAX_IL_TRANS	FIXED	7286	Maximum length of an IL transmission block.
MAX_BLKSIZE	FIXED	7294	Upper limit on block size of IL data sets.

(continued)

MIN_BLKSIZE	FIXED	1000	Lower limit on block size of IL data sets.
GMT_ADJUSTMENT	FIXED	8.0	Number of hours EARLIER than Greenwich to assume the local clock to be running. The value may be signed (for the Eastern hemisphere) and may carry the fraction ".0" or ".5" (for half-hour time zones).
MAXMATERS	FIXED	1	Limit on number of rematerizations of process before it stops.
NSWVOL	CHAR	'NSWP01'	Direct-access volume on which to create new data sets in NSW file space.
WSPVOL	CHAR	'NSWP01'	Direct-access volume on which to create new data sets in a tool workspace.

2.6. APPENDIX D: IL GRAMMAR

I. The File Transmission Protocol

```
<file-transmission>
        ::= <transmission-block> (1:n) <eot>
<transmission-block>
        ::= <byte-count> <block>
<byte-count>
        ::= unsigned two byte quantity greater than 0
<eot>
        ::= end of transmission.  a <byte-count> = 0
<block>
        ::= a block of file bytes, <byte-count> long
```

II. The Intermediate Language Grammar

```
<text-files>
        ::= concatenation <block> (1:n)
<text-files>
        ::= <text-file> "251"
          | <subfiles>  "251"   (not supported)
<subfiles>
        ::= <subfile> (1:p)     (not supported)
<subfile>
        ::= <text-file> "250"   (not supported)
```

III. File Records

```
<text-file>
        ::= <record> (0:q)
<record>
        ::= <data-record>
<data-record>
        ::= <rec-ctl> <key> (0:1) <item> (0:s)
```

(continued)

IV. Record control

```
<rec-ctl>
        ::= "224" | "225" | ... | "237"
         | "238" <n1>
         | "239" <n2>
         | "246"        new page
         | <format-eff>
<n1>
        ::= unsigned 8-bit quantity
<n2>
        ::= unsigned 16-bit quantity formed by
             concatenating two successive 8-bit bytes
<format-eff>
        ::= "242"   (line feed)
          | "243"   (vertical tab)
          | "244"   (form feed)
```

V. Keys

```
<key>                ::= <char>  (k)
```

VI. Data Record Items

```
<item>
        ::= <string>
          | <repeat>
          | <fill>
          | <special-character>
<string>
        ::= <str-len> <char> (0:r)
<repeat>
        ::= <rep-len> <char>
<str-len>
        ::= "0" | "1" | "2" | ... | "127"
<fill-len>
        ::= "128" | "129" | ... | "191"
<rep-len>
        ::= "192" | "193" | ... | "223"
<special-character>
          | "240"    backspace
          | "241"    horizontal tab
          | "245"    carriage return
          | "247" <ASCII-ctl>
<ASCII-ctl>
        ::= an ASCII control character
```

(continued)

VII. IL Types

PATTERN	VALUE RANGE	MEANING
0xxxxxxx	0 - 127	string length 0 : 127
10xxxxxx	128 - 191	fill length 0 : 63
110xxxxx	192 - 223	repeat length 0 : 31
1110xxxx	224 - 237	begin new record, advancing 0 - 13 records
11101110	238	begin new record, advancing 0 - 255 records
11101111	239	begin new record, advancing 0 - 65535 records
11110xxx	240 - 247	format effectors
	240	backspace
	241	horizontal tab
	242	line feed
	243	vertical tab
	244	form feed
	245	carriage return
	246	new page
	247	ASCII special character
111110xx	248 - 251	subfiles and blocking
	248	reserved
	249	reserved
	250	end of sub-file
	251	end of IL transmission
111111xx	252 - 255	reserved

REFERENCES

1) Cashman, Faneuf, and Muntz, "File Package: The File Handling Facility for the National Software Works". Document CADD-7612-2711, Massachusetts Computer Associates, Wakefield, Massachusetts, Revised December 27, 1976.

2) Schantz and Millstein, "The Foreman: Providing the Program Execution Environment for the National Software Works". Document CADD-7701-0111, Massachusetts Computer Associates, Wakefield, Massachusetts, January 1, 1977.

3) Ludlam, "PL/PCP -- An NSW Procedure-Call Protocol Package for PL/I". UCLA/OAC document UCNSW-402, November 15, 1980.

4) Ludlam and Rivas, "PL/MSG -- An MSG Interface for PL/I". UCLA/OAC document UCNSW-401, November 15, 1980.

5) Braden, "PL/B8 -- A PL/I Interface Package for NSWB8". UCLA/OAC document UCNSW-403, November 15, 1980.

6) Braden and Ludlam, "An IP Server for NSW". UCLA/OAC Technical Report TR7, April 1, 1976.

7) Braden, "PLOXI -- A PL/I Interface to Exhange". UCLA/OAC document UCNSW-407, November 15, 1980.

8) Farrell and Ludlam, "The NSW Basic Copy Machine". UCLA/OAC document UCNSW-203, November 20, 1980.

PART III

FP/360 THE BASIC COPY MACHINE

This section is separately available
as UCLA document UCNSW-203

ILLUSTRATIONS

3. PART III: THE BCM

3.1. BCM FUNCTIONAL SPECIFICATIONS

3.1.1. PURPOSE AND CAPABILITIES

The Basic Copy Machine, or BCM, is a CALL'able program for copying and transforming a data stream. It was designed and developed as the main working component of the National Software Works (NSW) File Package program (reference 1, 2), and in all aspects its design facilitates its use in that environment. However, it may be useful in other environments as well, and this document is addressed to the general caller as much as the File Package implementor and maintainer.

The BCM may be useful in any situation where it is desired to copy a data set with certain attributes of data, encodement, and storage, into a data set with different attributes. The BCM is most useful when each of the two sets of attributes corresponds to a well-defined native file type that has already been assigned a "Global File Type" (GFT) name; however, the user may also enumerate elementary attributes. Sections of this document will list available GFT's, available elementary file attributes, and supported transformations.

The non-NSW caller will see some peculiarities as a result of the BCM's NSW orientation. Notable among these are:

* The BCM describes the formats of its data files in terms of the NSW's "Global File Type" (GFT) names. The general caller may use a File-Package-provided subroutine to convert such a type name into a set of elementary attributes, or he may specify elementary attributes himself. The latter approach considerably complicates use of the BCM.

* The parametric interface to the BCM is not compact or clean. Some parts of it will not concern the general caller.

* The BCM expects a cooperative and friendly caller which has done a reasonable job of consistency checking the BCM's input parameters. Thus the BCM is not forgiving of errors.

* The BCM's capabilities for cross-network copying of data streams are virtually unavailable to any caller except the NSW File Package.

* Many subroutine references can be left unresolved by the non-network caller.

Figure 1: Basic Structure of the BCM

```
  Data Regions:     Copy Functions:     Files:

*--------------*    ****************    000000000000000
| INPUT        |   *   GET         *<--0  INPUT     0
|   Region     |<--*  Function     *   0      File  0
*--------------*    ****************    000000000000000
             |
             |
             V
            ************************
           * Initial TRANSFORMATION *
           *       Function         *
            ************************
             |
             |
             V
*--------------*    ****************    0000000000000000
| INTERMEDIATE |-->* Secondary PUT  *   0  Secondary   0
| Region       |   *   Function     *-->0  OUTPUT file 0
*--------------*    ****************    0000000000000000
             |
             |
             V
            ************************
           * Final TRANSFORMATION   *
           *       Function         *
            ************************
             |
             |
             V
*--------------*    ****************    0000000000000000
| OUTPUT       |-->* Primary PUT    *   0  Primary     0
| Region       |   *   Function     *-->0  OUTPUT file 0
*--------------*    ****************    0000000000000000
```

(arrows indicate possible data record flow)

3.1.2. FILES

The basic structure of the BCM is shown in Figure 1. The program uses three data files, each of which has attributes associated with an NSW GFT name.

* There is always a primary input file. If this is to be a local data set, it is allocated to and read through file name (DDNAME) INPUT. It may also be a remote file on another NSW host.

* There is an optional primary output file. If this is to be a local data set, the BCM will create it (if it already exists the old copy may have to be deleted first) and write it through file name WSPOUT. It may also be a remote file on another NSW host. The data written through this file may be reformatted if the file is assigned a different GFT from the input file.

* There is an optional secondary output file. If it is used, it must be a local data set, and it must have the same GFT as the input file.

The BCM also uses routine MSGJOUR, of the PL/MSG subroutine package, to write informative messages to the user. This routine, unless instructed otherwise (see the documentation for PL/MSG), will write to QSAM output file MSGJOUR, if such a file is allocated. If it is not allocated, no harm is done. If the program is executing in the foreground, MSGJOUR will also, unless instructed otherwise, write its output to the controlling TSO terminal via TPUT. There is an option to also write the output to another TSO terminal, if the named userid is logged on (reference 3).

Since the BCM is written in PL/I (for the IBM Optimizing Compiler), it can conceivably write diagnostic messages from the PL/I running system. These normally require a file named SYSPRINT. It is advisable to allocate such a file just in case of errors.

3.1.3. DATA TRANSFORMATIONS

The primary output file emits the data from the input file after applying any required transformations. These may include:

* Transformation into a compressed form. The compression scheme used is that defined for NSW Intermediate Language (IL); however, EBCDIC to ASCII translation can be suppressed, and the various ASCII-oriented format effectors that can be expressed in IL are never generated by the BCM.

* Expansion from the compressed form. The various ASCII-oriented format effectors that can be expressed in IL can be expanded or converted to EBCDIC control characters. ASCII to EBCDIC translation can be suppressed.

* Dimensional conversion among:

  (a) Two-dimensional data -- lines or records.

  (b) Three-dimensional data -- lines organized into pages.

  (c) Four-dimensional data -- overprinted records organized into lines, and optionally further organized into pages.

* Generating, stripping, or interpreting ASA carriage control. Note that when processing carriage control or format effectors of any kind, the BCM must make an initial assumption about the virtual position of the output file before any positioning is performed. This is always assumed to be on the non-existent line just preceeding the first line of the data space, with the horizontal position left undefined.

* Generating, stripping, or moving sequence number fields.

* Truncating or folding long records or pages.

* Forcing upper case.

* Stripping trailing fill characters (blanks for text data, binary zeros for binary data).

* Changing the OS/360 RECFM, LRECL, and BLKSIZE.

3.1.4. OPERATING ENVIRONMENTS

The BCM executes under IBM OS/360 MVT, in the environment provided by the PL/I Optimizing Compiler's execute-time library. The BCM can operate in either the foreground or the background.

* FOREGROUND OPERATION -- The BCM is most at home operating in the foreground, under the TSO Terminal Monitor Program (TMP). File allocation operations are handled by the TSO DAIR routine, through UCLA's PLIDAIR package (reference 4).

* BACKGROUND OPERATION -- The BCM will operate in the OS/360 background environment provided appropriate substitutes for the PLIDAIR entries named ALLOC, CREALL, FREE, and DELETE are provided. If file allocation operations are handled external to the BCM (as through Job Control Language statements), these may be stubs that return zero status codes. Renamed copies of routine FMNULL can be used (see the section entitled USEFUL SUPPORT ROUTINES).

The BCM can operate as either a local copy machine or a network copy machine, with some restrictions in the latter mode.

* LOCAL OPERATION -- If the input and output streams of the BCM are local data sets, then its use is essentially unrestricted. The caller will need to set certain parametric data to prescribed values in order to avoid actuating the network communications machinery. He may delete certain BCM subroutines that effect such communication, if the BCM is to be included in his load module directly.

* NETWORK OPERATION -- If either the input or the output of the BCM is not a local data set, then the BCM performs certain restricted network procedure calls. These calls will fail unless the BCM caller has already established himself as a legitimate NSW process of class "FLPKG". Further details of this mode of operation can be found in the section entitled "SPECIAL REQUIREMENTS FOR NETWORK COPIES".

3.1.5. BCM FILES

The BCM's actual input and output streams may require the allocation of files named INPUT (the input stream), WSPOUT (the primary output stream), and NSWOUT (the secondary output stream). Other functions use files named SYSPRINT (PL/I diagnostics), MSGJOUR (optional journaling output), and PARMS (optional parameters -- see the section entitled FPINIT -- ALTER DEFAULT VALUES).

3.2. BCM USER'S MANUAL

This section is addressed to the BCM user -- that is, to the programmer who is writing a program to call the BCM.

3.2.1. CALLING THE BCM

Conceptually, the BCM is called as:

```
BCM  ( input file description,
       primary output file description,
       secondary output file description,
       environment description,
       default values table )
 --> ( error description,
       primary output data set description,
       secondary output data set description )
```

The actual PL/I call looks like this:

```
DECLARE FPCOPY ENTRY (POINTER, POINTER, POINTER, POINTER);
CALL FPCOPY ( ADDR (environment_descriptor),
              ADDR (input_file_descriptor),
              ADDR (primary_output_file_descriptor),
              ADDR (secondary_output_file_descriptor) );
%INCLUDE source_library (DDFAULT);
```

where results are returned in descriptors, and the "default values table" is provided as a static external data structure. The descriptors are described in the next sections.

Figure 2: The BCM Data Region Descriptor

```
3 REC_CONTROL,                    /* POINTS TO CURRENT REC.   */
  4 (TEXTAD, KEYAD) POINTER,
  4 (LNGTEXT, LNGKEY, SKIPS) FIXED BIN(15),
3 FILENAME CHAR(8) VAR,           /* DDNAME, TITLE OPTION.    */
3 REALFILE FILE,                  /* POINTS TO FILE CONSTANT */
3 PCD,
  4 HOST,
    5 NUMBER FIXED BIN(15),        /* NETWORK HOST ADDRESS.   */
    5 NAME CHAR(32) VAR,           /* HOST MNEMONIC NAME.     */
    5 FAMILY CHAR(32) VAR,         /* HOST FAMILY NAME.       */
    5 RELATION FIXED BIN(15),      /* LOCAL, FAMILY, FOREIGN */
  4 DIRECTORY CHAR(56) VAR,        /* FIRST PART OF NAME.     */
  4 FNAME CHAR(56) VAR,            /* SECOND PART OF NAME.    */
  4 PHYS POINTER,                  /* PHYSICAL ENCODE. INFO. */
  4 IL_ENCODED BIT(1),             /* TRUE --> COPY IN IL.    */
3 FUD,                             /* FILE USAGE DESCR:       */
  4 APPROX_BIT_COUNT FIXED BIN(31),  /* TOTAL IL SIZE.        */
  4 ACTUAL_BLOCKSIZE FIXED BIN(15),  /* TO BE USED            */
  4 BUFFER_SIZE FIXED BIN(15),       /* FOR ALLOCATION.       */
  4 PASSWORD  CHAR(32) VAR,        /* ACCESS PASSWORD         */
  4 DSNAME CHAR(56) VAR,           /* FULL DATA SET NAME.     */
  4 USAGE FIXED BIN(15),           /* OLD OR NEW              */
3 GLOBAL_TYPE CHAR(32) VAR,        /* GLOBAL TYPE NAME        */
3 TYPE_DESCRIPTOR,
  4 TYPE_TYPE FIXED BIN(15),       /* NATIVE/FAMILY/FOREIGN   */
  4 GFD,                           /* GLOBAL FILE DESCR:      */
    5 CLASS FIXED BIN(15),         /* 1-->TEXT, 2-->BINARY.   */
    5 KEYLENGTH FIXED BIN(15),     /* 0 --> NO KEYS.          */
    5 DIMENSION FIXED BIN(15),     /* 1 TO 4.                 */
    5 BYTESIZE FIXED BIN(15),      /* USUALLY 8 ...           */
    5 HTAB,                        /* HORIZONTAL TABS...      */
      6 (ILCHAR, EBCCHAR) CHAR(1),
      6 (INCREMENT, STOPCOUNT, STOPS(20)) FIXED BIN(15),
    5 VTAB,                        /* VERTICAL TABS...        */
      6 (ILCHAR, EBCCHAR) CHAR(1),
      6 (INCREMENT, STOPCOUNT, STOPS(20)) FIXED BIN(15),
    5 LF,                          /* LINEFEED...             */
      6 (ILCHAR, EBCCHAR) CHAR(1),
      6 (INCREMENT, STOPCOUNT, STOPS(20)) FIXED BIN(15),
    5 FF,                          /* FORMFEED...             */
      6 (ILCHAR, EBCCHAR) CHAR(1),
      6 (INCREMENT, STOPCOUNT, STOPS(20)) FIXED BIN(15),
```

(Continued)

Figure 2 (continued): The BCM Data Region Descriptor

```
 4 LFD,                       /* LOCAL FILE DESCR:      */
  5 DIMENSIONAL_PREFERENCE (4) FIXED BIN(15),
  5 TEXT_LNG,
   6 (MAX, MIN) FIXED BIN(15),
  5 COMP_FACTOR FIXED BIN(15),
  5 KEY_OFFSET  FIXED BIN(15),
  5 FOLD_MARGIN FIXED BIN(15),
  5 PAGE_DEPTH FIXED BIN(15),
  5 HANDLING_OF,
   6 (KEYS, LONG_RECORDS, SHORT_RECORDS, LONG_PAGES,
      SHORT_PAGES, HTAB, VTAB, LF, FF, BSP, CR) FIXED BIN(15),
  5 OPTIONS,
   6 (FORCE_UPPER, SUPPRESS_TRANSLATE, SUPPRESS_EXPAND,
      KEEP_FILLS, INPUT_ONLY) BIT(1) ALIGNED,
 4 PSD,                       /* DEFAULT PHYS STRUCTS.  */
  5 DSORG CHAR(6) VAR,        /* DATA SET ORGANIZATION. */
  5 RECFM CHAR(6) VAR,        /* RECORD FORMAT.         */
  5 OPTCD CHAR(6) VAR,        /* OPTION CODES.          */
  5 LRECL FIXED BIN(15),      /* LOGICAL RECORD LENGTH. */
  5 BLKSIZE,                  /* BLOCKSIZE.             */
   6 (MAX, MIN) FIXED BIN(15),
  5 KEYLEN FIXED BIN(15),     /* RECORDED KEY LENGTH.   */
  5 RKP FIXED BIN(15),        /* RECORDED KEY OFFSET.   */
  5 SPACE_ALLOCATION,
   6 (PRIMARY, SECONDARY, PDSDIR) FIXED BIN(15),
3 VOLUME CHAR(6) VAR,         /* FOR CREATING NEW DS'S. */
```

3.2.1.1. BUILDING THE INPUT FILE DESCRIPTOR

The BCM input file descriptor is one of the BCM "Data Region Descriptors", as defined by %INCLUDE segment DDFILE and illustrated in Figure 2. This structure is used both as input parameters and as internal working storage by the BCM. Which fields must be preset by the caller depends on whether the input is remote or local.

3.2.1.1.1. FOR LOCAL INPUT

* HOST.RELATION -- Set to zero (the local system).

* DIRECTORY and FNAME -- These two fields, when concatenated with a period between them, form the local DSNAME, which must be fully qualified, but not quoted, and without a member name or generation number. The data set thus identified must already exist. Neither field should be blank or null.

* IL_ENCODED -- Set this bit to '0'B unless the input file is already encoded in NSW IL compressed form.

* GLOBAL_TYPE -- Set this to one of the NSW GFT names. Since this is a local file, this string will normally begin with "360-".

* GLOBAL_TYPE_DESCRIPTOR -- Set this entire substructure by calling routine FPDGTYP (see the section entitled USEFUL SUPPORT ROUTINES), passing the address of the entire file descriptor. Field GLOBAL_TYPE must have been already set before calling FPDGTYP. If the value of GLOBAL_TYPE does not begin with "360-", then follow the instructions for remote input files, just below.

* PASSWORD -- Processing of this field is deferred for now. Set it to a null string.

3.2.1.1.2. FOR REMOTE INPUT

If the BCM caller is dealing with a remote input file, it is assumed that he has access to the NSW data structures normally used to describe such files.

* HOST.NUMBER, .DIRECTORY, .FNAME, .PHYS, and .IL_ENCODED -- Set these fields directly from the PCD of the remote file copy selected.

* HOST.NAME and .FAMILY -- These fields are only used to format error messages, and they can be set to null strings. You can set them more aesthetically by calling routine MSGHTYP (see the section entitled USEFUL SUPPORT ROUTINES), passing HOST.NUMBER.

* HOST.RELATION -- Set this to 1 (family host) if the remote host is another 360-compatible system, or to 2 (foreign host) otherwise. If you have called routine MSGHTYP, you can look for the string "360" in field HOST.FAMILY to decide this. When in doubt, it is safe to assume a foreign host type.

* PASSWORD -- Set this field to whatever text string serves as a security key to permit access to the remote file. If the remote system does not use passwords, set this field to a null string. In normal File Package operation, this datum will have been given the File Package by its caller.

* GLOBAL_TYPE -- Set this to the NSW GFT name associated with the remote file copy.

* GLOBAL_TYPE_DESCRIPTOR -- Set this entire structure by calling routine FPDGTYP (see the section entitled USEFUL SUPPORT ROUTINES), passing the address of the entire file descriptor. field GLOBAL_TYPE must have been already set before calling FPDGTYP. If the string in GLOBAL_TYPE does not begin "360-", then you must also set all values of substructure GFD (either before or after calling FPDGTYP). Most of these fields can be copied directly from the GFD of the NSW file being accessed. The exceptions are the four fields named EBCCHAR. These should contain the EBCDIC codes for Tab (decimal 5), Vertical Tab (11), Line Feed (37), and Form Feed (12). If you must generate GFD information yourself, see the section entitled DEFINING FILE ATTRIBUTES for assistance.

3.2.1.2. BUILDING THE PRIMARY OUTPUT FILE DESCRIPTOR

The BCM primary output file descriptor is one of the BCM "Data Region Descriptors", as defined by %INCLUDE segment DDFILE and illustrated in figure 2. This structure is used both as input and output parameters and as internal working storage by the BCM. Which fields must be preset by the caller depends on whether the primary output is remote, local, or null.

3.2.1.2.1. FOR LOCAL PRIMARY OUTPUT

* USAGE -- set to 3 (a transformed output file).

* HOST.RELATION -- Set to zero (the local system).

* DIRECTORY and FNAME -- These two fields, when concatenated with a period between them, form the local DSNAME, which must be fully qualified, but not quoted, and without a member name or generation number. If DIRECTORY is null, the value of WSP_DIRECTORY in the default values table will be used. If FNAME is null, the value of WSP_DSN_PAT in the defalut values table will be used. However it is generated, the resulting DSNAME may contain up to 7 "?" characters. These will be replaced by pseudo-randomly chosen alphanumeric (not alphabetic) characters to generate a unique name (different substitutions will be tried until the generated name does not match any existing data set).

  If the data set name contains wild characters, and if every possible substitution for those characters yeilds a name which matches that of an existing data set, then the copy operation will fail. However, if the name does not contain wild characters, and if a data set of that name already exists, the BCM will delete the old copy and recreate it according to the attributes associated with this copy operation.

* IL_ENCODED -- Set to '0'B unless the primary output file is to be encoded in NSW IL compressed form.

* GLOBAL_TYPE -- Set this to one of the NSW GFT names. Unless IL_ENCODED is '1'B, the name chosen must begin with "360-".

* GLOBAL_TYPE_DESCRIPTOR -- Set this entire structure by calling routine FPDGTYP, passing the address of the entire file descriptor. Field GLOBAL_TYPE must be already set before calling FPDGTYP. If the value of GLOBAL_TYPE does not begin with "360-", then follow the instructions below for remote output files.

- PASSWORD -- Processing of this field is deferred for now. Set it to a null string.

3.2.1.2.2. FOR REMOTE PRIMARY OUTPUT

If the BCM caller is dealing with a remote primary output file, it is assumed that he has access to the NSW data structures normally used to describe such files.

* USAGE -- set this field to 3 to indicate a transformed output file.

* HOST.NUMBER -- Set this field to the NSW host number of the remote host. Normally, this will have been learned from an incoming FP-SENDME call from that host.

* HOST.NAME and .FAMILY -- These fields are only used to format error messages, and they can be set to null strings. You can set them more aesthetically by calling routine MSGHTYP (see the section entitled USEFUL SUPPORT ROUTINES), passing HOST.NUMBER.

* HOST.RELATION -- Set this to 1 (family host) if the remote host is another 360-compatible system, or to 2 (foreign host) otherwise. If you have called routine MSGHTYP, you can look for the string "360" in field HOST.FAMILY to decide this. When in doubt, it is safe to assume a foreign host type.

* PASSWORD -- Set this field to whatever text string serves as a security key to permit access to the remote file. If the remote system does not use passwords, set this field to a null string. In normal File Package operation, this datum will have been given the File Package by its caller.

* GLOBAL_TYPE -- Set this to the NSW GFT name to be associated with the remote file copy.

* GLOBAL_TYPE_DESCRIPTOR -- Set this entire structure by calling routine FPDGTYP (see the section entitled USEFUL SUPPORT ROUTINES), passing the address of the entire file descriptor. Field GLOBAL_TYPE must have been already set before calling FPDGTYP. If the string in GLOBAL_TYPE does not begin "360-", then you must also set all values of substructure GFD (either before or after calling FPDGTYP). Most of these fields can be copied directly from the GFD of the NSW file being accessed. The exceptions are the four fields named EBCCHAR. These should contain the EBCDIC codes for Tab (decimal 5), Vertical Tab (11), Line Feed (37), and Form Feed (12). If you must generate GFD information yourself, see the section entitled DEFINING FILE ATTRIBUTES for assistance.

* IL_ENCODED -- Set this bit to '1'B.

3.2.1.2.3. FOR NULL PRIMARY OUTPUT

If no primary output is desired, simply set USAGE to zero (null usage).

3.2.1.3. BUILDING THE SECONDARY OUTPUT FILE DESCRIPTOR

The BCM secondary output file descriptor is one of the BCM "Data Region Descriptors", as defined by %INCLUDE segment DDFILE and illustrated in figure 2. This structure is used both as input and output parameters and as internal working storage by the BCM. The output must be either null or to a local data set.

3.2.1.3.1. FOR LOCAL SECONDARY OUTPUT

* USAGE -- set this field to 2 to indicate an untransformed copy of the input file.

* HOST.RELATION -- Set to zero to indicate the local system.

* DIRECTORY and FNAME -- These two fields, when concatenated with a period between them, form the local DSNAME, which must be fully qualified, but not quoted, and without a member name or generation number. If DIRECTORY is null, the value of NSW_DIRECTORY in the default values table will be used. If FNAME is null, the value of NSW_DSN_PAT in the defalut values table will be used. However it is generated, the resulting DSNAME may contain up to 7 "?" characters. These will be replaced by pseudo-randomly chosen alphanumeric (not alphabetic) characters to generate a unique name (different substitutions will be tried until the generated name does not match any existing data set).

  If the data set name contains wild characters, and if every possible substitution for those characters yeilds a name which matches that of an existing data set, then the copy operation will fail. However, if the name does not contain wild characters, and if a data set of that name already exists, the BCM will delete the old copy and recreate it according to the attributes associated with this copy operation.

* IL_ENCODED -- Copy this datum from the input file descriptor.

* GLOBAL_TYPE and GLOBAL_TYPE_DESCRIPTOR -- Copy these data from the input file descriptor.

* PASSWORD -- Processing of this field is deferred for now. Set it to a null string.

3.2.1.3.2. FOR NULL SECONDARY OUTPUT

If no secondary output is desired, simply set USAGE to zero (null usage).

3.2.1.4. BUILDING THE ENVIRONMENT DESCRIPTOR

The BCM Environment Descriptor is a structure of the form defined by %INCLUDE segment DDPCPFMT and illustrated in Figure 3. The primary purpose of this descriptor is to communicate the status of the NSW Procedure Call Protocol (PCP) environment (reference 5) to those portions of the BCM which must do network communications. It is also used to communicate an error code and string back to the BCM caller. Only the latter function is of concern if the copy is to be local.

3.2.1.4.1. FOR LOCAL COPY OPERATIONS

If the BCM is operating only on local datasets, only two fields are of concern.

* ERROR_TYPE -- preset this field to zero.

* ERROR_STRING -- preset this field to a null string.

3.2.1.4.2. FOR NETWORK COPY OPERATIONS

If the BCM is operating on any remote file, then the caller must be using the PL/PCP package (reference 5).

* MASTER_ECB -- this word must have been passed to PCBEGIN as the PCP master ECB.

* LOCAL_PROCESS -- Store the local process handle returned from PCBEGIN here.

* CALLER -- Store the process handle of the calling process here.

* CALL -- Store the transaction handle of the call being executed here.

* ERROR_TYPE -- preset this field to zero.

* ERROR_STRING -- preset this field to a null string.

Figure 3: The BCM Environment Descriptor

```
2 MASTER_ECB FIXED BINARY(31), /* FOR TPEXAM CALLS          */
2 LOCAL_PROCESS POINTER,       /* TO A PROCESS STRUCTURE    */
2 CALLER        POINTER,       /* TO REMOTE PROCESS STRUCT  */
2 CALL          POINTER,       /* TO TRANSACTION STRUCTURE  */
2 CALL_TYPE     CHAR(6),       /* FROM TPEXAM               */
2 TERM_RECEIVED BIT(1),        /* MSG'S "TS" SETS THIS      */
2 ERROR_TYPE FIXED BIN(15),    /* FROM "FPERRNO"            */
2 REPLY_STRING CHAR(255) VAR,  /* PROCEDURE RESULTS         */
2 ERROR_STRING CHAR(255) VAR,  /* ONLY VARIABLE PART...     */
2 WORKB8,                      /* PL/B8 WORK AREA & RC      */
  3 B8_RETURN_CODE FIXED BIN(31),
  3 WORKAREA (35) FIXED BIN(31);
```

Figure 4: The BCM Default Values Table

```
DECLARE 1 DDFAULT STATIC EXTERNAL,
          2 MSG_TIMEOUT   FIXED BINARY(31) INIT (60000),
          2 PCP_TIMEOUT   FIXED BINARY(31) INIT (1800000),
          2 IMP_TIMEOUT   FIXED BINARY(31) INIT (60000),
          2 MAXCOPIES FIXED BIN (15) INIT (1),
          2 MAXMATERS FIXED BIN (15) INIT (1),
          2 GMT_ADJUSTMENT FIXED BIN(15,4) INIT(8.0),
          2 MAX_BLKSIZE FIXED BIN(15) INIT(7294),
          2 MIN_BLKSIZE FIXED BIN(15) INIT(1000),
          2 MAX_IL_TRANS FIXED BIN(15) INIT(7286),
          2 (NSW_DIRECTORY INIT('AHA179.NSW'),
            NSW_DSN_PAT   INIT('GEN.NSW?????'),
            WSP_DIRECTORY INIT('AHA179.NSW'),
            WSP_DSN_PAT   INIT('GEN.WSP?????'))
                            CHAR(44) VAR,
          2 GENERICNAME  CHAR(16) VAR INIT('FLPKG'),
          2 NSWVOL CHAR(6) INIT('NSWP01'),
          2 WSPVOL CHAR(6) INIT('NSWP01'),
          2 VERIFY_IMPORT BIT(1) INIT(0B);
```

3.2.1.5. BUILDING THE DEFAULT VALUES TABLE

The BCM Default Values Table is a structure of the form defined by %INCLUDE segment DDFAULT and illustrated in Figure 4. You must include this structure somewhere in the load module that calls the BCM. %INCLUDE segment DDFAULT will set default values in the structure; however, your program can alter any of these prior to calling the BCM. You can also call routine FPINIT (see the section entitled USEFUL SUPPORT ROUTINES), which will execute a "GET DATA" on the entire structure, thus allowing execute-time changes by the program user. The structure is designed to provide default values for the NSW File Package program, so many of the fields are not used by the BCM. Those which are are:

* PCP_TIMEOUT -- If the BCM must issue an FP-SENDME call to another network host, this value is used for the PCCALL timeout.

* MAX_BLKSIZE and MIN_BLKSIZE -- These values are used as the bounds in selecting a device-dependent block size for a local output data set to hold IL compressed data.

* MAX_IL_TRANS -- The maximum length of a network transmission block.

* NSW_DIRECTORY -- The string to be used if the DIRECTORY field of the secondary output descriptor is null.

* NSW_DSN_PAT -- The string to be used if the FNAME field of the secondary output descriptor is null. This should contain at least one, but not more than seven, "?" characters.

* WSP_DIRECTORY -- The string to be used if the DIRECTORY field of the primary output descriptor is null.

* WSP_DSN_PAT -- The string to be used if the FNAME field of the primary output descriptor is null. This should contain at least one, but not more than seven, "?" characters.

* NSW_VOLUME -- The name of the local filespace volume on which secondary output data sets are to be created.

* WSP_VOLUME -- The name of the local filespace volume on which primary output data sets are to be created.

Figure 5: BCM Error Codes

type:	string:	meaning:
0	--	No error.
4	--	One of the BCM File Descriptors is invalid or inconsistent.
9	--	An internal error has prevented the encoding of a network message -- a BCM bug is probably indicated.
14	--	An I/O error occurred in reading or writing one of the BCM files. Network and local device errors are treated the same.
15	--	The BCM is transmitting a file already encoded in IL to a remote host. A record in the input file is longer than the value of MAX_IL_TRANS in the Default Values Table.
16	GFT name	A record to be transformed in to the indicated GFT format has a data portion greater than LFD_TXT_LNG.MAX, and the value of LFD.HANDLING_OF.LONG_RECORDS indicates that this is an error.
17	GFT name	The input file does not have keys, and the output GFT indicates in field LFD.HANDLING_OF.KEYS that the output file must have non-generated keys.
19	DSNAME	An output file descriptor specifies values of DIRECTORY and FNAME that map into the name of an existing data set. If there are "?" characters in the name, all possible permutations and combinations of alphanumeric substituends yeild such duplicate names.
21	DSNAME	An I/O error occurred during data set creation.
22	DSNAME	There is no file space which both has sufficient room to creat the output data set, and is legally accessible by the File Package.
23	DSNAME	An unknown error occurred during data set creation.

(Continued)

Figure 5 (Continued): BCM Error Codes

26	Host name	The BCM is negotiating with the remote host which owns its input file. The negotiating network transaction failed, probably due to a timeout as specified by PCP_TIMEOUT or MSG_TIMEOUT of the Default Values Table.
27	--	The BCM is negotiating with the remote host which owns its input file. That host has returned an unintelligible message.
28	--	The network connection to a remote host cannot be opened by PL/MSG. If the PL/MSG LOG option is being used, the messages written to file MSGJOUR may have more information.
29	--	The network connection to a remote host cannot be closed by PL/MSG. If the PL/MSG LOG option is being used, the messages written to file MSGJOUR may have more information.

3.2.1.6. RETRIEVING BCM RETURNED VALUES

The BCM returns values in several places in its parameter structures:

3.2.1.6.1. PRIMARY ERROR CODE

Field ERROR_TYPE of the Envirnonment descriptor will contain zero if the copy completed normally, and non-zero otherwise. Field ERROR_STRING will be either null or a string suitable for inserting into an error message. Figure 5 lists possible error codes, their meanings, and suggested error messages with insertion points for the variable string.

3.2.1.6.2. GENERATED DATA SET DESCRIPTORS

If no error occurred, the primary and secondary output file descriptors will have certain fields filled in to describe any <u>local</u> data sets actually created from within the BCM. Note that if you fool the BCM into thinking that it has created a data set when the data set was actually created previously, these values will be meaningless. See the section entitled FMNULL -- BYPASSING PLIDAIR for more information on this mode of operation.

* DSNAME -- This field will contain the fully qualified, quoted, data set name actually assigned the output data set. All "?" characters will have been replaced by alphanumerics.

* ACTUAL_BLKSIZE -- This field will contain the block size actually selected for the data set.

* PSD -- All subfields of this substructure except for the BLKSIZE substructure indicate the actual values used to create the data set.

3.2.2. USEFUL SUPPORT ROUTINES

Several routines of various NSW-related packages can be used by the BCM caller to some advantage:

3.2.2.1. FPDGTYP -- DEFINE GLOBAL TYPE

Routine FPDGTYP fills in substructure TYPE_DESCRIPTOR based on the value of field GLOBAL_TYPE (see Figure 2). It is called by:

```
DECLARE FPDGTYP ENTRY (POINTER, FIXED BIN (15));
CALL FPDGTYP ( ADDR (file_descriptor), error_code);
```

where:

"file_descriptor" is a BCM "Data Region Descriptor", as defined by %INCLUDE segment DDFILE and illustrated in Figure 2. Field GLOBAL_TYPE must already be filled in.

"error_code" is returned as zero after a normal operation or as non-zero if GLOBAL_TYPE contained a native but unknown GTF name.

Three cases exist:

(1) If "error_code" is non-zero, then the GFT is illegal. nothing else will have been filled in.

(2) If "error_code" is zero and field TYPE_TYPE is zero, then the GFT is a known native type, and all fields of TYPE_DESCRIPTOR have been filled in.

(2) If "error_code" is zero and field TYPE_TYPE is non-zero, then the GFT is an unknown type, and has been assummed to be valid but non-native. Substructure GFD has not been altered, and the rest of TYPE_DESCRIPTOR has been filled in with values that describe the preferred method of storing non-native IL-encoded data sets on the 360 system. Before calling the BCM, you must fill in all values of GFD (either before or after calling FPDGTYP). In the normal network-transfer case, the BCM caller will already be holding GFD data received from his caller. For other cases, refer to the section entitled DEFINING FILE ATTRIBUTES for assistance.

FPDGTYP uses File Package subroutines FPERRNO and FPGTTAB. FPGTTAB is loaded dynamically, so it must be made available through one of the mechanisms available to program fetch (previously loaded, Linkpack, Linklib, JOBLIB, STEPLIB, Task library, etc.).

3.2.2.2. FPINIT -- ALTER DEFAULT VALUES

Routine FPDGTYP can be called to allow modification of the default values table by the program user. It is called by:

```
DECLARE FPINIT  ENTRY (FILE, *),
        PARMS   STREAM INPUT FILE;
%INCLUDE source_library (DDFAULT);
CALL FPDGTYP ( PARMS, DDFAULT );
```

FPINIT issues a "GET DATA" against its input file, which it opens to file name PARMS, and its input structure, which it calls P. Thus an example of valid file input to FPINIT might be:

```
//PARMS  DD *
 P.PCP_TIMEOUT = 40000,
 P.WSPVOL = 'WKSPC2' ;
```

FPINIT will function normally with or without the semicolon normally used to terminate data-directed input. It will act as a no-operation routine if no PARMS file is allocated.

3.2.2.3. FMNULL -- BYPASSING PLIDAIR

FMNULL is an Assembler subroutine that merely locates its last (or returning) parameter cell, stores a halfword of zeros there, and returns. Thus it can be substituted for any entry of either of the forms:

```
DECLARE  routine  ENTRY ( ... )
                  RETURNS (FIXED BIN (15));

     or

DECLARE  routine  ENTRY ( ... , error_code),
         error_code  FIXED BIN (15);
```

Since all entries of the PLIDAIR package are of this form, FMNULL can be substituted at Linkage-Edit time for entries ALLOC, CREALL, FREE, and DELETE. This makes the BCM bypass all file-allocation operations. That, it turn, makes the BCM executable in a batch environment, where its files are defined by JCL statments of the form:

```
//INPUT   DD  <define the input file>
//WSPOUT  DD  <define the primary output file>
//NSWOUT  DD  <define the secondary output file>
```

However, the BCM caller must understand that the data set names and characteristics that are returned by the BCM in its file descriptors are not meaningful in this case.

3.2.2.4. MSGHTYP -- FINDING HOST RELATIONSHIPS

When the BCM is operating in its network mode, it is necessary to classify the remote host system. Routine MSGHTYP can do this. It is a part of the NSW PL/MSG subroutine package, and is thus documented elsewhere (reference 3). Parsimoniously put, its calling sequence is:

```
DECLARE  MSGHTYP ENTRY (FIXED BIN (15),
                   CHAR (32) VAR, CHAR (32) VAR);

DECLARE  1 file_descr,
           %INCLUDE source_library (DDFILE);

CALL MSGHTYP (file_descr.HOST.NUMBER,  /* you preset */
              file_descr.HOST.NAME,    /* returned   */
              file_descr.HOST.FAMILY); /* returned   */
```

3.2.3. DEFINING FILE ATTRIBUTES

The simplest way to use the BCM, and the way that is recommended, is to assign a complete set of attributes to each file in the form of an NSW GFT name. These names are then converted to a set of file attributes by subroutine FPDGTYP (see the section entitled USEFUL SUPPORT ROUTINES). If no GFT name exists that properly defines your file type, it may be that such a name should be added; see local NSW maintenance personnel.

However, if your requirements are unique, then you can build the entire file attribute descriptor substructure yourself. You begin by constructing a BCM Data Region Descriptor using %INCLUDE packet DDFILE, illustrated in Figure 2. Fields GLOBAL_TYPE and TYPE_DESCRIPTOR.TYPE_TYPE are not examined by the BCM (except that GLOBAL_TYPE may be moved into ERROR_STRING of the Environment Descriptor under certain circumstances) so you need not set them; however, most of substructures GFD (the Global File attribute Descriptor), LFD (the Local File attribute Descriptor), PSD (the Physical Structure Descriptor) must be set.

It is important to understand that values for certain attributes are often required by the BCM even when they do not appear to have meaning for the particular data type. This is because they may be needed to perform a type conversion into that type from a non-native type about which no attributes are known. For example, a "page depth" is tabulated for a two-dimensional data type if conversion of non-native three-dimensional data into that type could occur, even though page depth has no meaning for a two-dimensional file.

* GFD.CLASS -- This field determines whether the data consists of characters (CLASS = 1) or binary bytes (CLASS = 2), with the following consequences:

  (1) Character-class data represents an array of ASCII graphics of dimensionality between 1 and 4 (see the discussion of dimensionality below). A full complement of format effectors is defined for use in positioning graphics within the array. Unspecified array positions are assumed to contain the fill character "blank", which is also used for optimal compression in IL.

  Data with dimensionality of 2 or higher is organized into "records", which may include character-string keys. A common use of these "keys" will be to record "sequence numbers" associated with text lines by some text editors, compilers, and other such programs.

(2) Binary-class data is of dimensionality 1 or 2, representing either a single byte string or a sequence of (short) byte strings called "records", respectively. There are no format effectors other than record separators. In two-dimensional data, a record may include a character-string key as well as the binary text.

For binary-class data, the "fill" character used for optimum compression in IL is a byte of binary zero.

* GFD.KEYLENGTH -- This field declares the width of the key (or sequence number) field associated with each data record. A value of zero means that no key field exists. Keys are always character strings, even in binary-class files.

* GFD.DIMENSION -- This field declares the file data to represent 1, 2, 3, or 4 dimensions. This concept is defined as follows:

(1) (At this writing, the BCM does not yet support one-dimensional files.) One-dimensional data consists of a stream of bytes (or characters) that are not logically grouped into lines or records. The single dimension corresponds to file size, and is effectively unbounded.

```
{ BYTE [ c ],    c= 1 to file_size }
```

For character-class files, regular horizontal format effectors (see Figure 2) are possible, but no other format effectors would be meaningful. The data may be broken arbitrarily into record-like strings for convenience in handling, but it is understood that these strings are not logical records. A one-dimensional file cannot have keys.

(2) Two-dimensional data consists of a stream of bytes (or characters) divided into records or lines. Keys are permitted, and if they appear there is a key included with each record. The first dimension is bounded by the "Record Length Range" datum of the LFD, but the second corresponds to file size, and is effectively unbounded.

```
{  KEY [ k, r ], BYTE [ c, r ],

     for:  c= 1 to max-record-text-length,
           k= 1 to key-width,
           r= 1 to record-count,          }
```

For character-class files, it is possible to define any kind of regular horizontal format effectors and regular vertical interval format effectors, but no other kinds are meaningful.

(3) Three-dimensional data consists of a stream of characters, grouped into lines, which are then grouped into pages. The first dimension is bounded by the "Record Length Range" datum of the LFD, and the second by the "page depth" datum, but the third corresponds to file size, and is effectively unbounded.

```
{  KEY [ k, r, p ], BYTE [ c, r, p ],

     for:   c= 1 to max-record-text-length,
            k= 1 to key-width,
            r= 1 to page-depth,
            p= 1 to page-count              }
```

Only character-class data can be three-dimensional, and it is meaningful to define all regular format effectors. Keys are legal, but will probably be rare.

(4) Four-dimensional data consists of a stream of characters, grouped into records, which are then grouped into lines, which may then be grouped into pages. The first dimension is bounded by the "Record Length Range" datum of the LFD, and the second corresponds to overprinting and is unbounded. The third dimension is bounded by "page depth", while the fourth corresponds to file size and is also unbounded.

```
{  KEY [ k, r, l, p ], BYTE [ c, r, l, p ],

     for:   c= 1 to max-record-text-length,
            k= 1 to key-width,
            r= 1 to max-overprint-depth,
            l= 1 to page-depth,
            p= 1 to page-count             }
```

Only text-class data can be four-dimensional, and it is meaningful to define all regular and irregular format effectors. Keys are legal, but will probably be rare.

* GFD.BYTESIZE -- The BCM can only process files consisting of 8-bit bytes, so this datum should be set to 8.

* GFD.HTAB, .VTAB, .LF, and .FF -- These substructures tell the BCM how to expand the ASCII-type format effectors that can be expressed in IL-compressed text. Whether these instructions are used at all depends on the settings of switches in substructure LFD.HANDLING_OF, described later.

The names stand for Horizontal Tab, Vertical Tab, Line Feed, and Form Feed, respectively. The functions performed by these format effectors are handled in other ways in a 360-compatible system, and these other ways also have IL representations. Therefore, the BCM itself never generates such format effectors when doing IL compression, and the only time it can encounter these forms is when it is expanding a file that was compressed by another NSW host system. If you cannot be sure that format effectors will not be encountered, but you wish them to have minimum significance, then set these values this way (all values are decimal):

	substructure:			
field:	HTAB	VTAB	LF	FF
ILCHAR	241	243	242	244
EBCCHAR	5	11	37	12
INCREMENT	1	1	1	1
STOPCOUNT	0	0	0	0

Otherwise, three cases can exist for each format effector. The ILCHAR and EBCCHAR fields will always be set the same, but the values of the other fields vary as follows:

(1) If the format effector is to remain undefined, set both INCREMENT and STOPCOUNT to zero. In this case, if an occurrence of the format effector is found during the copy operation, it will be replaced by its EBCDIC equivalent, the value of EBCCHAR.

(2) If the format effector is defined to have regularly spaced "stops" in column (line) 1 and in every "n" columns (lines) thereafter, set INCREMENT to "n" and STOPCOUNT to zero.

(3) If the format effector is defined to have a specific set of "stops", set INCREMENT to zero, STOPCOUNT to the number of such stops (maximum of 20), and vector STOPS to the (1-origin) values of the stops themselves. Unused elements of STOPS need not be initialized. If, during the copy operation, an occurrence of the format effector is found to lie beyond the column (line) indicated by the last used element of STOPS, it will be replaced by its EBCDIC

equivalent, the value of EBCCHAR.

* LFD.DIMENSIONAL_PREFERENCE -- This datum defines a preference ordering of the four dimensionalities for situations where dimensional conversion may be required. It consists of a vector of four numbers, corresponding to dimensionalities 1 through 4. Each number consists of the sum of two parts: First, an indication of whether conversion from that dimension is permitted, with values:

  (000) Used for the dimension which requires no conversion.

  (256) Permit the conversion from this dimension.

  (512) Permit this conversion only if interactive user approval can be obtained.

  (768) Forbid conversion from this dimension.

  Second, a preference part simply ranking this dimensionality in terms of its relative preference compared to another dimensionality with the same first-part. This part can take values from 0 to 255. For example, if:

```
DIMENSIONAL_PREFERENCE (1) = 768;  /* Forbid             */
DIMENSIONAL_PREFERENCE (2) =   0;  /* No conversion      */
DIMENSIONAL_PREFERENCE (3) = 257;  /* permit and prefer  */
DIMENSIONAL_PREFERENCE (4) = 258;  /* prefer only after 3 */
```

  Then the order of preference in selecting a file to transform into this file type is 2 (most preferred), 3, and 4 (least preferred), with conversion from dimesionality 1 strictly forbidden.

* LFD.TXT_LNG.MAX and .MIN -- These two fields give the maximum and minimum number of bytes of data (exclusive of keys) that a record of this type can contain. This must be compatible with GFD.KEYLENGTH, PSD.RECFM, and PSD.LRECL.

* LFD.COMP_FACTOR -- Set this field to the estimated IL-compressibility of typical data of this type, as an integer percentage. For example, "75" would mean that an IL file of this type typically occupies 3/4 as much disk space as its uncompressed equivalent.

* LFD.KEY_OFFSET -- Set to the 1-origin index of the data byte before which the sequence number field appears in the record. For example, "1" means leading sequence numbers, and "73" would position the sequence numbers as for a standard IBM card-image file.

* LFD.FOLD_MARGIN -- If it is ever necessary to break a long input record into two output records to comply with LFD.TEXT_LNG.MAX, the continuation record may be offset after a leading blank field. Set this datum to the 1-origin column in which continuation data is actually to begin. For example, "1" will not offset the continuation record at all.

* LFD.PAGE_DEPTH -- Set this field to the number of lines (not records) to a printer page.

* LFD.HANDLING_OF.KEYS -- How to generate missing keys. Values are:

  (0) Simply delete the key field.

  (1) Blank fill the key field.

  (2) Generate sequence numbers by counting records.

  (3) The file must have a real, information-bearing key field. Refuse to process an input file type that does not.

* LFD.HANDLING_OF.LONG_RECORDS -- How to handle input records with more data bytes than permitted by LFD.TEXT_LNG.MAX. Values are:

  (0) Simply truncate the record.

  (1) Fold the record -- that is, make a continuation record beginning in the column specified by LFD.FOLD_MARGIN.

  (2) Call the situation an unrecoverable error.

* LFD.HANDLING_OF.SHORT_RECORDS -- How to handle input records with fewer data bytes than permitted by LFD.TEXT_LNG.MIN. Values are:

  (0) Pad the record with fill characters.

  (1) Try to ignore the problem.

* LFD.HANDLING_OF.LONG_PAGES -- How to handle input pages with more lines than permitted by LFD.PAGE_DEPTH. Values are:

  (0) Truncate the page.

(1) Fold the page -- that is, make it into two pages.

(2) Ignore the problem.

* LFD.HANDLING_OF.SHORT_PAGES -- How to handle input pages with fewer lines than required by LFD.PAGE_DEPTH. Values are:

(0) Pad out the page with blank lines.

(1) Ignore the problem.

* LFD.HANDLING_OF.HTAB -- How to expand the IL encodement of the ASCII Horizontal Tab format effector, if it is encountered in the input file. Values are:

(0) Substitute an EBCDIC Horizontal Tab code.

(1) Expand according to the GFD.HTAB substructure of the BCM's Input File Descriptor.

(2) Expand according to the GFD.HTAB substructure of the BCM's Primary Output File Descriptor.

* LFD.HANDLING_OF.VTAB -- How to expand the IL encodement of the ASCII Vertical Tab format effector, if it is encountered in the input file. Values are:

(0) Substitute an EBCDIC Vertical Tab code.

(1) Expand according to the GFD.VTAB substructure of the BCM's Input File Descriptor.

(2) Expand according to the GFD.VTAB substructure of the BCM's Primary Output File Descriptor.

* LFD.HANDLING_OF.LF -- How to expand the IL encodement of the ASCII Line Feed format effector, if it is encountered in the input file. Values are:

(0) Substitute an EBCDIC Line Feed code.

(1\ Expand according to the GFD.LF substructure of the BCM's Input File Descriptor.

(2) Expand according to the GFD.LF substructure of the BCM's Primary Output File Descriptor.

* FFD.HANDLING_OF.FF -- How to expand the IL encodement of the ASCII Form Feed format effector, if it is encountered in the input file. Values are:

(0) Substitute an EBCDIC Form Feed code.

(1) Expand according to the GFD.FF substructure of the BCM's Input File Descriptor.

(2) Expand according to the GFD.FF substructure of the BCM's Primary Output File Descriptor.

* LFD.HANDLING_OF.BSP -- How to expand the IL encodement of the ASCII Backspace format effector, if it is encountered in the input file. Values are:

(0) Substitute an EBCDIC Backspace code.

(1) Interpret as a destructive backspace order. That is, delete the previous character, if there is one.

(2) Interpret as a non-destructive backspace order. That is, move the "cursor" back one position, if that is possible, but do not delete any characters, and do not shorten the output record.

* LFD.HANDLING_OF.CR -- How to expand the IL encodement of the ASCII Carriage Return format effector, if it is encountered in the input file. Values are:

(0) Substitute an EBCDIC Carriage Return code.

(1) Interpret as a destructive carriage return order. That is, delete all characters previously placed in the current record.

(2) Interpret as a non-destructive carriage return order. That is, move the "cursor" back one to the beginning of the current line, but do not delete any characters, and do not shorten the output record.

* LFD.OPTIONS.FORCE_UPPER -- If this bit is set, the characters of the data are to be translated into all upper case.

* LFD.OPTIONS.SUPPRESS_TRANSLATE -- If this bit is set, then IL data of this type is to be considered to be already in EBCDIC.

* LFD.OPTIONS.SUPPRESS_EXPAND -- If this bit is set, then this type represents data the clear-text format of which is the same as the IL-compressed format. This bit is not processed by the BCM.

* LFD.OPTIONS.KEEP_FILLS -- If this bit is set, then trailing fill characters in records are considered to be significant. Otherwise, they can be stripped without loss of information.

* LFD.OPTIONS.INPUT.ONLY -- If this bit is set, then data declared to be of this type is to be considered to be read-only. This bit is not processed by the BCM.

* PSD.DSORG -- the recommended data set orgainization for creating new files. The BCM only supports physical sequential data, so this field should contain the value "PS".

* PSD.RECFM -- the recommended record format. The BCM supports:

```
F[B[S]][A][T] (fixed-length records)
V[B][S][A][T] (variable-length records)
U[A][T]       (undefined-length records)
```

* PSD.OPTCD -- Data management option codes. The BCM merely copies these into the data set label, so anything acceptable to OS/360 is acceptable here.

* PSD.LRECL -- The logical record length. This must be compatible with the values of GFD.KEYLENGTH and LFD.TXT_LNG.

* PSD.BLKSIZE.MAX and .MIN -- The bounds on physical block size. The BCM may choose a value within this range which is compatible with PSD.RECFM and PSD.LRECL, and which optimizes space utilization on the selected physical device. If the two values are the same, then no variation in block size is allowed.

* PSD.KEYLEN -- Length of random-access retrieval key. The BCM merely copies this into the data set label. No random retrieval is ever done in the BCM itself.

* PSD.RKP -- The offset of the random-access retrieval key within a record. The BCM merely copies this into the data set label. No random retrieval is ever done in the BCM itself.

* PSD.SPACE_ALLOCATION.PRIMARY -- The recommended initial space allocation value, in selected blocksize units, to be used if no better file-size data is available.

* PSD.SPACE_ALLOCATION.SECONDARY -- The recommended incremental space allocation value, in selected blocksize units.

* PSD.SPACE_ALLOCATION.PDSDIR -- The recommended number of 256-byte blocks to allocate for a partioned data set (PDS) directory. Since the BCM does not yet support PDS's, this value should be zero.

3.2.4. SPECIAL REQUIREMENTS FOR NETWORK COPIES

In order to use the BCM for network operations, the caller must be familiar with the operation of an NSW File Package, and with the mechanisms for implementing such a process at UCLA. He must have already materialized an MSG process of class "FLPKG", using the PL/PCP subroutine package (reference 5). The BCM environment descriptor must accurately describe the PL/PCP environment (see the section entitled BUILDING THE ENVIRONMENT DESCRIPTOR).

If the BCM input is remote, then the BCM will invoke procedure FP-SENDME of any FLPKG process on the indicated remote host, using routines PCCALL and PCEXAM.

If the BCM primary output is remote, then the BCM will assume that its caller is responding to an FP-SENDME procedure call from the indicated remote host, and that field CALL of the environment descriptor is a handle on that transaction. The BCM will then complete the transaction using routine PCREPLY.

Theoretically, the BCM can copy a remote input file to a remote output file; however, such a request is never generated in the NSW system, so in practice, it will not be explicitly supported.

3.3. BCM MAINTENANCE MANUAL

This section is addressed to those responsible for updating and maintaining the BCM package. It describes the logical flow of the machine and the logical responsibilities of the various submodules.

3.3.1. BASIC STRUCTURE

The basic structure of the BCM is illustrated in Figure 1. It consists of a machine with three external files, one input and two output, and three internal "Data Regions", each associated with one of the files. During BCM execution, data records move from the input file, through each of the internal Data Regions in turn, and to the output files. Movement of data between an external file and an internal data region is accomplished by a GET or PUT routine. Movement of data between internal data regions is accomplished by a TRANSFORMATION routine. Collectively, these routines are known as Copy Functions.

The current implementation of the BCM is represented in Figure 6, which is another aspect of Figure 1. Figure 6 should be interpreted somewhat like a wiring diagram. The direction of data record flow is from top to bottom, and each line is marked with one or more of the three possible record formats: IL, CT (Clear Text), or NT (Normalized Text).

Conceptually, the BCM sports a variety of mode switches by which it can be parameterized. Using one set of switches, the BCM can be caused to perform one of at least three basic copy types (local copy, remote get, or remote send). Similar switches control conversion of data among the clear-text, normalized, and IL-encoded forms. Switches, marked by upper-case names in figure 6, direct flow to, from, and through various active processing components. Each of these has the property that for every record accepted, an unpredictable number of output records, including zero, may be produced.

These notions divide the switches into two types:

Figure 6: BCM Parameterization Switches

```
          **********************    *********************
          *                    *    *                   *
          *     LOCAL_GET      *    *    NETWORK_GET    *
          *                    *    *                   *
          **********************    *********************
         (IL or CT) |                         | (IL or CT)
                    *-------->0 <*> 0<--------*
                                 |   SOURCE_IS_LOCAL switch
                                 |
                 *<-----------0 <*> 0  INPUT_FORMATTING_TYPE switch
            (CT) |                  |
         *****************          |
         *               *          V KEEPING_AN_NSW_COPY switch
         *  NORMALIZER   *          *-> 0------------>*
         *               *          |                 |
         *****************          |      (IL or CT) |
           (NT)  |                  |         ******************
                 *----------------->*         *  LOCAL PUT     *
                                    |         *                *
                                    |         ******************
                                    |
                                    V
                                 0 <*  WRITING_AN_OUTPUT switch
                                 |
                                 | OUTPUT_FORMATTING_TYPE switch
                                 |
       *<-------------------0   <*>   0------------------->*
       |               *<-----0  V  0---->*                |
       |               |         0        |                |
       |               |         |        |                |
 (IL)  |         (CT)  |         |  (IL)  |        (NT)    |
 **************  **************  |  *************  ****************
 *  IL_       *  * IL_        *  |  * IL_       *  *  NORMALIZED  *
 *  EXPANDER  *  * COMPRESSER *  |  * REBLOCKER *  *  TEXT        *
 *            *  *            *  |  *           *  *  EXPANDER    *
 **************  **************  |  *************  ****************
 (CT)  V         (IL)  V         V  (IL)  V        (CT)    V
       *-------------->*-------->*<-------*<---------------*
                                 |
                                 |  (IL or CT)
                                 |
                              t  |  f   OUTPUT_IS_LOCAL switch
                     *--------0 <*> 0---------*
                     |                        |
          **********************   **********************
          *                    *   *                    *
          *     LOCAL_PUT      *   *    NETWORK_PUT     *
          *                    *   *                    *
          **********************   **********************
```

3.3.1.1. DATA ROUTING SWITCHES

These switches are concerned with getting data into and out of the BCM. Their settings are primarily determined by the basic type of copy operation being executed, and by the types of the external data sets to be processed. These switches are:

* the SOURCE_IS_LOCAL switch

* the KEEPING_A_SECONDARY_COPY switch

* the WRITING_AN_OUTPUT switch

* the OUTPUT_IS_LOCAL switch

At the same time these switches are set, and based upon the same parametric data, each data region is assigned a set of "file data attributes".

3.3.1.2. DATA CONVERSION SWITCHES

These switches are concerned with moving data across data region boundaries. Their settings are primarily determined by the relationships between the file data attributes assigned the two regions invloved. These switches are:

* the INPUT_FORMATTING_TYPE switch

* the OUTPUT_FORMATTING_TYPE switch

Conceptually, both of these switches can be set to perform any type of data conversion; however, in the present implementation, most potential values of the INPUT_FORMATTING_TYPE switch are undefined. This results in the restriction that the data written through the secondary output file must be an untranslated, unrecoded copy of the input stream.

It must be understood that the switches described above are abstractions for the purpose of describing the logic of the machine. In fact, the functions of the switches are implemented by a generator mechanism which dynamically binds an appropriate subroutine into the machine at the point where the switch has its effect. The BCM is a dynamically bound collection of routines, consisting of a root or control routine and five slots to be filled with one of a number of candidate processing modules (see Figure 7). FPCOPY, the control routine which receives control when the BCM is called, is responsible for the selection and control of the other modules. From the values set in the three file descriptors (discussed in the section entitled "CALLING THE BCM"), FPCOPY is able to select five subroutines appropriate to fill in the five variable slots for the particular invocation of the BCM, and then to use these routines to set up a copy operation, to perform the copy, and to clean up the entire operation. BCM operation is thus composed of four phases:

* the Generator phase

* the Resource Allocation phase

* the Work phase

* the Resource Freeing phase

More will be said about these phases in a the section entitled PROCESSING PHASES.

Figure 7: The Generated Copy Machine:

```
                  Resource      Work          Resource
                   Allocation    Entries:       Freeing
                    Entries:                     Entries:
                *-------------*-------------*--------------*
GET             |             :             :              |
 Copy           |      (parametrically selected routine)   |
  Function      |             :             :              |
                *-------------*-------------*--------------*
Initial         |             :             :              |
 Transform      |      (parametrically selected routine)   |
  Copy Function |             :             :              |
                *-------------*-------------*--------------*
Secondary       |             :             :              |
 PUT Copy       |      (parametrically selected routine)   |
  Function      |             :             :              |
                *-------------*-------------*--------------*
Final           |             :             :              |
 Transform      |      (parametrically selected routine)   |
  Copy Function |             :             :              |
                *-------------*-------------*--------------*
Primary         |             :             :              |
 PUT Copy       |      (parametrically selected routine)   |
  Function      |             :             :              |
                *-------------*-------------*--------------*
```

3.3.2. DATA STRUCTURES

3.3.2.1. DATA RECORDS

There are three forms in which a data record can exist for the purposes of the BCM.

3.3.2.1.1. FORMATTED CLEAR TEXT

Non-IL records are stored on a 360 disk in a format called "formatted clear text", or simply "clear text". Interpretation of a clear text record generally requires a full knowledge of the Local File Attributes (LFA's) and Physical Storage Attributes (PSA's) for its type; therefore, only records of a 360-native type can exist in clear text form.

A clear text record consists of a single string of bytes. If the RECFM field of its LFD contains the letter "A", each record will begin with a single byte of ASA carriage control; no other type of format effector is defined. If the record has keys, the key-length field of the GFD and the key-offset of the LFD together define a substring of the record which is the key; the remaining bytes constitute the record text field.

3.3.2.1.2. NORMALIZED TEXT

Normalized Text is an IL-like internal representation for a record which is used by FP/360 when converting from one type to another. This representation consists of the triple:

(<skipcount>, <key>, <text>)

where,

* <skipcount> is the amount of vertical movement from the preceding line to the current one; for example, zero means that the current line is to overprint the preceding line. There is also a reserved value of <skipcount> meaning "form feed".

* <key> is the isolated key string; and

* <text> is a string containing the characters or bytes of the text field less any insignificant trailing fill bytes.

A record in this representation may be considered to be independent of any of the local host-specific mappings defined by the LFA's and PSA's.

3.3.2.1.3. IL TRANSMISSION BLOCKS

An IL Transmission Block (reference 2) is the form in which data is moved between Network hosts of the NSW system. The block consists of a string of transmission bytes of a specified bit width (but the BCM will process only 8-bit bytes). This string consists of a two-byte binary count field followed by the indicated number of data bytes. The data bytes consist of a catenation of data records, each containing an encodement of the triple (<skipcount>, <key>, <text>). The encodement is such that <skipcount> and <key> are easily extractable; however, <text> is compressed according to the standard NSW data compression grammar (reference 2), and its length can only be determined by parsing in accord with that grammar.

3.3.2.2. THE DATA REGION DESCRIPTOR

A file descriptor, or Data Region Descriptor, has a dual personality, as can be seen from Figure 2. It represents the characteristics of an actual input or output data set (local, remote, or null) to which it is connected through one of the GET or PUT Copy Functions. It also represents a set of internal data characteristics which, together with another file descriptor to which this one is connected through one of the TRANSFORMATION Copy Functions, defines a set of data conversions for that Copy Function to implement.

More specifically, the Data Region Descriptor contains information of the following sorts:

* Descriptors of the current record residing in this Data Region.

* Descriptors of the PL/I FILE allocated to the GET or PUT function associated with this Data Region.

* Descriptors of the Network file allocated to the GET or PUT function associated with this Data Region.

* The "Global Type" name associated with the data as it is represented when it passes through this region, along with an expanded descriptor defining just what attributes that type name implies.

Because of the great variability in the operations that may actually go on in moving data into and out of a Data Region, the Data Region Descriptor includes many data elements which will be used only in specific cases. Still, there are always three full descriptors attached to a BCM execution, regardless of the paucity of relevant information that some of them may contain in some cases. The generator phase of the BCM will associate with a Data Region only those Copy Function routines which are compatible with the information actually present in the region.

Note that the Data Region Descriptor does not include any actual data record buffers, since the need for and size of these varies widely according to the selected Copy Function routines. Accordingly, buffers are managed by, and belong to, the Copy Function subroutines themselves.

3.3.2.2.1. FILE AND RECORD CONTROL

The following fields of the Data Region Descriptor (see Figure 2) are those that Copy Functions use to manipulate PL/I files and data buffers. Except as noted, this means that the BCM caller need not concern himself with them; their values before, during, and after execution are not of interest to him.

* TEXTAD -- points to the data string which represents the current data record associated with this Data Region Descriptor, or, in the case of normalized internal form, to the data portion of the record. This address may be null (or invalid) when no such string exists.

* KEYAD -- the address of the extracted key field of a normalized internal data record.

* LNGTEXT -- is the length of the current data string (if any) pointed to by TEXTAD.

* LNGKEY -- is the length of the current key field (if any) pointed to by KEYAD.

* SKIPS -- is an accumulator for the "skip count" component of a normalized internal data record.

* FILENAME -- ddname used to allocate local files. Currently the BCM is hard-wired (in FPCOPY) to always use INPUT, NSWOUT, and WSPOUT for the three possible files.

* REALFILE -- a PL/1 FILE variable which is set to one of the PL/I FILE constants INFILE, KPFILE, or OUTFILE. This is true even if the associated file is a network file instead of a local data set.

3.3.2.2.2. FILE USAGE DESCRIPTOR

The File Usage Descriptor contains data pertinent to this particular access to the data:

* APPROX_BIT_COUNT -- this value is used by the Resource Allocation Routine of the PUT Copy Function to allocate space for a local output file. It is determined by the Resource Allocation Routine of the GET Copy Function. If the input is a local file, it is calculated from the number of track extents allocated to the file, multiplied by the number of blocks that fit on that track, multiplied by the number of bits in a block of the file. If the input is from a foreign host, the file size is passed as part of the initial NETWORK connection to the Resource Allocation Routine, and the value is simply stored.

  AT present, this value remains constant for a particular copy operation, even through IL translation. Once it is determined for the input data region, the BCM control routine, FPCOPY, in a breach of discipline, copies the value to both the secondary and primary output data regions. To get the correct value for local IL data sets, it should be multiplied by the compression ratio during the Resource Allocation Routines of the translate Copy Functions, but because it must be done by all translate Resource Allocation Routines, and because a null routine is used for the Resource Allocation Routine in some cases, FPCOPY performs this function instead of a special routine which would do nothing else. This should be corrected in a future version.

* ACTUAL_BLOCKSIZE -- this value is set to the actual blocksize found in a local input file, or chosen for a local output file.

* BUFFER_SIZE -- the BUFFER_SIZE is the length of any dynamic buffer gotten for records in this data region. It is determined and set by the Resource Allocation Routine of the particular Copy Function that outputs into this region, and used by the Resource Freeing Routine to free storage. The value for local files is assigned from the LRECL field of the PSD (LRECL-4 for files of RECFM=V). For data files at foreign hosts, the BUFFER_SIZE is simply the agreed-upon transmission size, less 2 for the transmission control bytes.

* PASSWORD -- that data needed to gain access to the file on the host. The BCM currently ignores this field for local files. It will pass this value to foreign hosts for a NETWORK input request.

* DSNAME -- for non-local files, this field is ignored. For local files, this is a concatenation of PCD.DIRECTORY and PCD.FNAME, with a period between them, and with any "wild characters" ("?") replaced by whatever alphanumerics can be found to result in a unique ID for a local data set.

* USAGE -- the type of I/O. Obviously, only certain values of this datum are valid for certain files. At the present, the BCM only looks at the value for the output files. The values for USAGE are:

  * 0 --> "Dummy": the file represented by this Data Region Descriptor is not to be read or written.

  * 1 --> "Input": this Data Region Descriptor's file exists and is to be read.

  * 2 --> "NSW": this Data Region Descriptor's file will be written, and is to be created in the NSW File Space.

  * 3 -- "Local": this Data Region Descriptor's file will be written, but it is not to be created in the NSW File Space.

Figure 8: The Function Control Area

```
2 PCP_POINTER                   POINTER,
2 ALLOC_RESOURCE_ROUTINE        ENTRY,
2 FREE_RESOURCE_ROUTINE         ENTRY,
2 WORK_ROUTINE                  ENTRY,
2 INPUT_REGION                  POINTER,
2 OUTPUT_REGION                 POINTER,
2 LOCAL_STATUS,
  3 HAS_LOCALLY_INITIALIZED         BIT(1) ALIGNED,
  3 HAS_LOCALLY_FINALIZED           BIT(1) ALIGNED,
  3 HAS_BEEN_ALLOCATED              BIT(1) ALIGNED,
  3 HAS_TERMINATED              BIT(1) ALIGNED,
2 GLOBAL_STATUS_POINTER         POINTER
```

Figure 9: The Connection Descriptor

```
2 CONREQ,    /* DIRECT-CONNECTIONS... */
  3 CTYPE CHAR(4),
  3 CWIDTH FIXED BIN(15),
  3 CID FIXED BIN(15),
  3 CQDEPTH FIXED BIN(15),
2 CONCONTROL,
  3 CONHANDLE POINTER,
  3 CONCECB FIXED BIN(31),
  3 CONOECB FIXED BIN(31),
  3 CBPTR POINTER,
  3 CBLENGTH FIXED BIN(15);
```

3.3.2.3. THE FUNCTION CONTROL AREA

The function control area is illustrated in Figure 8. It represents one of the Copy Functions that move data into and out of the data regions. The values of its data fields are determined by FPCOPY based on the file and data characteristics of the pertinent Data Region Descriptors. There are five Function Control Areas, corresponding to the five Copy Functions (see figure 7). Once they are initialized, FPCOPY can use the same calling sequences for all possible BCM confiqurations, by using the values of appropriate fields of the Function Control Area. Those fields are:

* PCP_POINTER -- this points to the environment descriptor passed to the BCM by its caller. It is used to report error conditions and is also used by the Network routines.

* ALLOC_RESOURCE_ROUTINE -- The address of the selected Resource Allocation Routine.

* FREE_RESOURCE_ROUTINE -- The address of the selected Resource Freeing Routine.

* WORK_ROUTINE -- The address of the selected Work Routine.

* INPUT_REGION -- The address of the Data Region Descriptor from which this Copy Function takes its input records. For the GET Copy Function, this will be null.

* OUTPUT_REGION The address of the Data Region Descriptor to which this Copy Function gives its output records. For the PUT Copy Functions, this will be null.

* LOCAL_STATUS -- Status flags that communicate the Control function's status between calls to it and between it and FPCOPY. The bits are:

    * HAS_LOCALLY_INITIALIZED -- any initializations required on the first entry to the Work Routine have been completed.

    * HAS_LOCALLY_FINALIZED -- the Work Routine has cleaned up any local initializations and need not be called again.

    * HAS_BEEN_ALLOCATED -- the Resource Allocation Routine for this Copy Function has successfully completed.

  - * HAS_TERMINATED -- the output data of this invocation of the Work Routine has exhausted the input. Implicit in this status condition is a request for more input on the next call. FPCOPY will attempt to satisfy this by calling the previous Copy Function.

* GLOBAL_STATUS_POINTER -- the address of global status flags referenced by all Copy Functions. The bits pointed to are:

  - * CLEAN_UP_NEEDED -- local Copy Function finalizations must now be performed. This status flag is set only on the last sequence of calls to the Work Routines. This flag is set after any error or end-of-data condition has been found and processed.

  - * COPY_EOD -- the GET Copy Function has exhausted the input data. Preliminary clean-up can be performed.

  - * TRANSMISSION_ERROR -- an unrecoverable error has occurred in either a GET or PUT Copy Function. The BCM is to be aborted.

  - * TRANSLATION_ERROR -- an unrecoverable error has occurred in an EDIT Copy Function. The BCM is to be aborted.

  - * ALLOCATION_ERROR -- a local Copy Function initialization has failed. The BCM is to be aborted.

3.3.2.4. THE CONNECTION DESCRIPTORS

The two Connection Descriptors describe the two possible network connections that may possibly be used instead of the PL/I FILEs described in the Data Region Descriptors for the INPUT and OUTPUT regions. These descriptors logically overlay those file descriptors. They are not in the same control block only for historical reasons. The format of a Connection Descriptor is shown in Figure 9. We do not describe it in great detail here, as it is only of interest when the BCM is used in the context of the NSW File Package, and in that context it is self explanatory.

3.3.2.5. THE ENVIRONMENT DESCRIPTOR

The Environment Descriptor is illustrated in Figure 3, and described in the section entitled "CALLING THE BCM".

3.3.2.6. THE DEFAULT VALUES TABLE

The Default Values Table is illustrated in Figure 4, and described in the section entitled "CALLING THE BCM".

Figure 10: Basic Logic After Generation

```
WORK (i):
  | IF i < 6
  |    THEN | UNTIL F(i).HAS.TERMINATED
  |         |       | CALL F(i).WORK_ROUTINE
  |         |       | CALL WORK (i+1)


SETUP (i):
  | If i > 5
  |    THEN | CALL WORK (1)
  |
  |    ELSE | CALL F(i).RESOURCE_ALLOC_ROUTINE
  |         | IF no errors
  |         |    THEN | CALL SETUP (i+1)
  |         |         | CALL F(i).RESOURCE_FREE_ROUTINE


 CALL SETUP (1)
```

3.3.3. PROCESSING PHASES

BCM operation consists of four phases: The Copy-Function Generator phase, the Resource Allocation phase, the Work phase, and the Resource Freeing phase. Usually, these phases are executed in turn; however, at any time, in any subroutine of any Selectable Module, an unrecoverable error may occur. If this happens, FPCOPY is notified, an error message is generated, and FPCOPY initiates the proper sequence of exiting calls, enabling all routines to accomplish their needed de-allocations, before control is returned to the BCM caller. The actual shape of the logic governing execution of the BCM after generation is complete is illustrated in Figure 10.

3.3.3.1. THE GENERATOR PHASE

The five Copy Functions are represented by five Function Control Areas (see Figures 7, 9). The Generator phase consists of assigning Selectable Modules to each of these areas. This is done by examining the values in each Data Region Descriptor and selecting routines that are compatible with the file attributes and required transformations. At the end of this phase, the Function Control Areas have been initialized, and the machine has the appearance shown in Figure 7.

3.3.3.2. THE RESOURCE ALLOCATION PHASE

The Resource Allocation phase consists of executing the selected Resource Allocation routine indicated by each initialized Function Control Area. These routines are executed in a well-defined order, so that each can use information set by those preceding.

The functions of the Resource Allocation Routine of a Copy Function are to set information in the associated data regions, usually copying information from the input region to the output region, to open files, and to allocate dynamic buffers where they are indicated. The Resource Allocation Routine never processes actual file data. If the Resource Allocation Routine completes normally, its work will be undone by the related Resource Free Routine. If it completes abnormally, it must perform its own clean-up and indicate its failure. In such a case, FPCOPY will skip directly to the Resource Freeing phase, beginning at the point immediately <u>after</u> the return from the Resource Freeing Routine that corresponds the the Resource Allocation Routine that failed.

3.3.3.3. THE WORK PHASE

The Work phase copies the actual data records. It is executed only if the Resource Allocation Phase completed successfully. It consists of an iterated and structured sequence of calls to the work routines of the Copy Functions in sequence. Each function is called repeatedly until it indicates that it requires new input; then its predecessor is called to supply that input. Likewise, after each call to a function, if that call has produced output, the successor function is called to dispose of that output. In this way, each Copy Function can emit zero, one, or more output records for each input record absorbed. This scheme is general enough to encompass the management of routines that do various kinds of blocking, deblocking, absorbing of null records, etc.

When a Work Routine is called, the input string pointed to by the input Data Region Descriptor may be either:

- null (an invalid string pointer)
- a new input string
- the residue of the input string left from the previous call.

and when it exits, the input string is left as:

- null
- non-null, i.e. that portion of the original input string which is not reflected in the Work Routine's output string. If this is the case, the HAS_TERMINATED flag in the Copy Function's local status area will not be set and the Work Routine will be called again by FPCOPY with this input.

The output of a Work Routine on exit is either:

- null
- non-null, a new string. There is no concatenation performed on the output string pointed to by the output Data Region Descriptor, although many input strings may be condensed and stored in a buffer internal to the Work Routine itself. The output string will be null for each absorbed input until a complete output string is finally produced, completely flushing the internal buffer.

The actual internal mechanics of a specific Work Routine, particularly the translation operations, are highly dependent on the given attributes of the input and output Data Region Descriptors.

When a Work Routine is first entered, initializations have already been performed for the Copy Function by the Resource Allocation Routine; however, some initializations local to the work routine itself may need to be performed on the first entry. It is up to the designer of a Selectable Module to determine what initializations are massive enough to warrant inclusion in the separate routine. The Work Routine performs any local initialization and sets the local status flag, HAS_LOCALLY_INITIALIZED. It is then able to continue its normal operations. Note that during the final call to a Work Routine, indicated by the global status flag, CLEAN_UP_NEEDED, the Work Routine must undo all of its own initializations.

The Work Routine may encounter an end-of-data or error condition during its own internal operations. If so, it sets the appropriate global status flag, sets its output string to null, and exits. When a Work Routine sets such a global flag, FPCOPY will take appropriate action, including the setting of the global status flag CLEAN_UP_NEEDED. FPCOPY then repeats the nested calling of WORK routines one final time. Thus each Work Routine needs only test the global flag CLEAN_UP_NEEDED to determine if this is his last opportunity to perform local buffer flushing and finalization. Again, it is up to the designer of a Selectable Module to determine which finalizations are local to the Work Routine and which should be done in the Resource Freeing Routine.

3.3.3.4. THE RESOURCE FREEING PHASE

The Resource Freeing phase is entered after the Work phase is either complete or bypassed. It consists of a sequence of calls to the Resource Freeing routines of the Copy Functions.

The Resource Free Routine of a Copy Function is entered only if the corresponding Resource Allocation Routine was entered and completed successfully. This is independent of whether or not the Work Routine(s) were ever executed. The routine frees dynamic buffers, closes files, etc. Like the Resource Allocation Routine, the Resource Free Routine never processes file data.

3.3.4. SELECTABLE MODULES

A Selectable Module is the entity which is assigned to a Copy Function. The five Copy Functions are (see figure 7):

1) Input,

2) Initial transformation,

3) Secondary output,

4) Final transformation,

5) Primary output.

Each Selectable Module is made up of three distinct subroutines, corresponding to the three phases of the copy operation: Resource Allocation, Work, and Resource Freeing. These entries are defined to provide flexibility to the designer of a Selectable Module, who may, for instance, have to work around severe main-storage restrictions. Logically, though, they implement a single function, and so the routines that comprise them must be treated together.

Every subroutine of every Selectable Module is called by FPCOPY with the following:

```
CALL <function>.<phase-specific-entry-name>
     (ADDR (<function-control-area>),
      ADDR (<connection-control-block));
```

Because of the uniformity of these calls, it is possible to support the entire range of BCM subroutine combinations with one set of calls in FPCOPY (see figure 10). This is because the truly variable parameters to the routines are reflected in the Copy Function structure and in the Data Region Descriptors already.

3.3.4.1. TRANSFORMATIONAL COMPONENTS

Those components of the Basic Copy Machine which are responsible for converting the format or content of file data will now be described. In general, the source and target files have attributes which may be the same or different. In the latter case, the component implements conversion. Input records are not checked for conformity to dimensional constraints, but output records will always be correct. One exception must be noted: If the Global File Type of the input and output of the BCM are the same, all transformational components are short-circuited, and no attribute policing takes place.

3.3.4.1.1. NORMALIZER

The Normalizer component of the BCM accepts clear text records and produces normalized text records. It de-formats its input according to the input-LFD attributes. Skip counts are generated from ASA carriage control, if present, and/or completely null (text and key) records, or null records with duplicate keys.

3.3.4.1.2. NORMALIZED TEXT EXPANDER

The Normalized Text Expander component accepts normalized text records and produces clear text records. The output is formatted according to the ouput-LFD attributes. Skip counts are converted to ASA carriage control, if appropriate, and/or to records with blank text fields and possible null or duplicate keys.

3.3.4.1.3. IL COMPRESSOR

The IL Compressor accepts clear text records and produces compressed IL records. Input is deformatted according to the input-LFD attributes. Record controls are generated from ASA carriage control, if present, and/or completely null records, or null records with duplicate keys.

3.3.4.1.4. IL EXPANDER

The IL Expander accepts compressed IL records and produces clear text records. Output is formatted according to the outpuf-LFD attributes. Record controls are converted to ASA carriage control, if appropriate, and/or to records with blank text fields and possible null or duplicate keys.

3.3.4.1.5. IL REBLOCKER

The IL Reblocker will accept an IL transmission block and produce one or more IL transmission blocks of a different maximum transmission block size. This is required, for instance, when copying an IL-encoded file to a remote file package which cannot handle the blocksize in which the file already exists.

In the present implementation, the IL Reblocker is present only as a stub. This means that situations where reblocking might be required will be legal, but if any overlong block is actually encountered, it will be treated as an unrecoverable copy error.

Even in future versions, the BCM will never reblock IL to obtain a larger block size. Since reblocking can only be done by completely parsing all the compressed text, it is assumed that the inefficiencies of handling small blocks are less than the effort of reblocking. In fact, the reblocker will use the simplest of possible algorithms:

* It will pass blocks that are already short enough straight through.

* If, at a data record boundary during parsing of a long block, the unparsed residue becomes short enough to be legal output, the block will be broken at that point regardless of how short the parsed portion may be.

* Otherwise, the first break of a block will occur at the last IL record boundary before the one which would make the first fragment too long.

* If none of these conditions can be met, (if there exists a record longer than the IL blocksize) an error condition exists, and the copy machine will abort the entire procedure.

3.3.4.1.6. THE NULL TRANSFORMATION

Whenever a transformational function bridges two data regions with identical characteristics, so that no data editing is indicated, FPCOPY selects a null transformational routine. This routine merely copies the data pointers from its input region to its output region.

3.3.4.2. COPY MACHINE I/O COMPONENTS

The Basic Copy Machine has one input and two output streams. These are controlled by dynamically-selected components. In general, these components are not sensitive to the encodement of the records, which may be formatted clear text or IL transmission blocks. However, the present implementation will not support Network transmission of clear text records, so the Network-handling components will not be requested to process other than IL transmission blocks.

3.3.4.2.1. THE LOCAL GET FUNCTION

The local get function is switched in when the copy machine's input data comes from a local data set. The component's responsibilities include:

* Locating and acquiring control over the input data set.

* Opening the data set.

* Filling in any PSA's that thus become known, particularly the total filesize.

* Acquiring record buffers.

* Retrieving the data set's records sequentially until end-of-file.

* Releasing buffers.

* Closing and releasing the data set.

* Notifying the copy machine of any exceptional conditions that arise.

3.3.4.2.2. THE NETWORK GET FUNCTION

The Network get function is switched in when the copy machine's input data comes from a remote file package. The component's responsibilities include:

* The scheduling and completion of a SENDME procedure call, directed to the selected remote NSW File Package. Most of the arguments to this call were inputs to the procedure call that the BCM is itself executing. Exceptions are:

  1) The transmission bytesize is hardwired at 8 bits.

  2) The maximum transmission record size is taken from the Default Values Table.

  3) The "family information" parameter will be of PL/B8 type EMPTY to indicate that the transmission is to be in IL transmission blocks.

     Note: this is a restriction on the current implementation; in subsequent versions, clear text may be transmitted between 360 hosts.

* Recording the results of *SENDME for* later use by other components. Of particular interest will be the total file size.

* Opening a PL/MSG RECV connection (reference 3) according to the negotiations agreed on during SENDME.

* Acquiring record buffers.

* Retrieving the file's blocks sequentially until either the connection closes or some other component signals that an encoded end-of-file has been found.

* Releasing buffers.

* Closing the connection.

* Notifying the copy machine and the other NSW File Package of any exceptional conditions that arise.

3.3.4.2.3. THE LOCAL PUT FUNCTION

The local put function is switched in when any of the copy machine's output goes to a local data set. This is usually true during the "import", "export", and "transport" procedures. If two outputs are being produced, this component will be used twice, but with different parameters. For instance, one may be writing clear text and the other IL transmission blocks. In any case, the component's responsibilities include:

* Creating the output data set, and acquiring control over it. Notice that this may use size data saved by the selected GET function. If the data set name is fully specified, and if a data set of that name already exists, then the old copy will be deleted.

* Opening the data set.

* Writing records sequentially until signaled by the copy machine that no more remain.

* Closing and releasing the data set, or deleting it if there was an error.

* Notifying the copy machine of any exceptional conditions that arise.

3.3.4.2.4. THE NULL PUT FUNCTION

When one of the BCM's output streams is not to be produced, FPCOPY selects a null PUT routine. This routine does essentially nothing at all.

3.3.4.2.5. THE NETWORK PUT FUNCTION

The Network put function is switched in when the copy machine's primary output data goes to a remote host. This is the case when the BCM is executing an NSW SENDME procedure call. The component's responsibilities include:

* Replying to the SEND procedure being executed. The following data is returned:

  1) The connection identifier is hardwired as 1.

  2) The transmission bytesize is hardwired at 8 bits.

  3) The actual IL transmission block size is the minimum of: a) the size requested in the call; and b) a limiting value taken from the Default Values Table.

  4) The "family information" parameter will be of NSWB8 type EMPTY to indicate that the transmission is to be in IL transmission blocks.

     Note: This is a restriction in the current implementation; in subsequent versions, clear text transmission may be supported between 360 family hosts.

* Opening a PL/MSG SEND connection according to the negotiations agreed on.

* Transmitting the file's blocks sequentially until signaled by the copy machine that no more remain.

* Transmitting an in-band ending status mark to the remote File Package.

* Closing the connection.

* Notifying the copy machine of any exceptional conditions that arise.

3.4. APPENDIX -- AVAILABLE GFT'S

```
GFT=360-KEYPUNCH,                                          *
  RECFM=FB,                 (FB, VBA, ETC. ETC.)           *
  LRECL=80,                 (80, 121, 137, ETC.)           *
  BLKSIZE=(400,6400),       ((MIN, MAX) OR FIXED)          *
  CMPFAC=40,                (EST IL/DISK BYTES * 100)      *
  KOFFS=73,                 (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=1,                (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                    (NATIVE DIMENSION)             *
  UC=Y,                     FORCE UC: (YES, NO)            *
  KEYHDL=B,                 (REQ, GEN, BLANK, NO)          *
  LONGR=F,                  (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                 (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                  (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                 (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                 (YES, NO)                      *
  CLASS=T,                  (TEXT, BINARY)                 *
  LNGKEY=8,                 (NOT IN THE DA SENSE)          *
  TXTLNG=72,                TEXT LENGTH RANGE              *
  DIM1=FORBID,              DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                             *
  DIM3=(ASK,1),                                            *
  DIM4=(ASK,2)


GFT=360-PRINT,                                             *
  RECFM=VBA,                (FB, VBA, ETC. ETC.)           *
  LRECL=137,                (80, 121, 137, ETC.)           *
  BLKSIZE=(141,4000),       ((MIN, MAX) OR FIXED)          *
  SPACE=(010,02),           (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=50,                (EST IL/DISK BYTES * 100)      *
  KOFFS=0,                  (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=1,                (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=3,                    (NATIVE DIMENSION)             *
  UC=N,                     FORCE UC: (YES, NO)            *
  KEYHDL=N,                 (REQ, GEN, BLANK, NO)          *
  LONGR=F,                  (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                 (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=F,                  (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                 (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                 (YES, NO)                      *
  CLASS=T,                  (TEXT, BINARY)                 *
  LNGKEY=0,                 (NOT IN THE DA SENSE)          *
  TXTLNG=(1,132),           TEXT LENGTH RANGE              *
  DIM1=FORBID,              DIMENSIONAL PREFERENCES        *
  DIM2=(PERMIT,2),                                         *
  DIM3=(PERMIT,1),                                         *
  DIM4=(ASK,3)
```

```
GFT=360-BINARY,                                           *
  RECFM=VBS,               (FB, VBA, ETC. ETC.)           *
  LRECL=0,                 (80, 121, 137, ETC.)           *
  BLKSIZE=(1,4000),        ((MIN, MAX) OR FIXED)          *
  SPACE=(005,01),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=60,               (EST IL/DISK BYTES * 100)      *
  KOFFS=0,                 (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=1,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=N,                    FORCE UC: (YES, NO)            *
  KEYHDL=N,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  INONLY=N,                (YES, NO)                      *
  CLASS=B,                 (TEXT, BINARY)                 *
  KPFILL=Y,                (TRAIL FILLS ARE SIGNIF; Y, N) *
  SUPRXL=Y,                (PERFORM CODE TRANS; Y, N)     *
  LNGKEY=0,                (NOT IN THE DA SENSE)          *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=FORBID,                                            *
  DIM4=FORBID
```

```
GFT=360-LOAD,                                             *
  RECFM=U,                 (FB, VBA, ETC. ETC.)           *
  LRECL=0,                 (80, 121, 137, ETC.)           *
  BLKSIZE=(1,20000),       ((MIN, MAX) OR FIXED)          *
  SPACE=(010,10,20),       (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=100,              (EST IL/DISK BYTES * 100)      *
  KOFFS=0,                 (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=0,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=N,                    FORCE UC: (YES, NO)            *
  KEYHDL=N,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  INONLY=N,                (YES, NO)                      *
  CLASS=B,                 (TEXT, BINARY)                 *
  KPFILL=Y,                (TRAIL FILLS ARE SIGNIF; Y, N) *
  SUPRXL=Y,                (PERFORM CODE TRANS; Y, N)     *
  LNGKEY=0,                (NOT IN THE DA SENSE)          *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=FORBID,                                            *
  DIM4=FORBID
```

```
GFT=360-PLI-CARDS,                                        *
  RECFM=FB,                (FB, VBA, ETC. ETC.)           *
  LRECL=80,                (80, 121, 137, ETC.)           *
  BLKSIZE=(80,4000),       ((MIN, MAX) OR FIXED)          *
  SPACE=(005,01),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=40,               (EST IL/DISK BYTES * 100)      *
  KOFFS=73,                (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=1,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=Y,                    FORCE UC: (YES, NO)            *
  KEYHDL=B,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  LNGKEY=8,                (NOT IN THE DA SENSE)          *
  TXTLNG=72,               TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=(ASK,1),                                           *
  DIM4=(ASK,2)
```

```
GFT=360-PLI-CC-CARDS,                                     *
  RECFM=FB,                (FB, VBA, ETC. ETC.)           *
  LRECL=80,                (80, 121, 137, ETC.)           *
  BLKSIZE=(80,4000),       ((MIN, MAX) OR FIXED)          *
  SPACE=(005,01),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=40,               (EST IL/DISK BYTES * 100)      *
  KOFFS=73,                (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=2,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=Y,                    FORCE UC: (YES, NO)            *
  KEYHDL=B,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  LNGKEY=8,                (NOT IN THE DA SENSE)          *
  TXTLNG=72,               TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=(ASK,1),                                           *
  DIM4=(ASK,2)
```

```
GFT=360-PLI-SOURCE,                                       *
  RECFM=VB,                (FB, VBA, ETC. ETC.)           *
  LRECL=104,               (80, 121, 137, ETC.)           *
  BLKSIZE=(104,4000),      ((MIN, MAX) OR FIXED)          *
  SPACE=(005,01),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=50,               (EST IL/DISK BYTES * 100)      *
  KOFFS=1,                 (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=1,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=Y,                    FORCE UC: (YES, NO)            *
  KEYHDL=B,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  LNGKEY=8,                (NOT IN THE DA SENSE)          *
  TXTLNG=(1,92),           TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=(ASK,1),                                           *
  DIM4=(ASK,2)
```

```
GFT=360-PLI-CC-SOURCE,                                    *
  RECFM=VB,                (FB, VBA, ETC. ETC.)           *
  LRECL=104,               (80, 121, 137, ETC.)           *
  BLKSIZE=(104,4000),      ((MIN, MAX) OR FIXED)          *
  SPACE=(005,01),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=50,               (EST IL/DISK BYTES * 100)      *
  KOFFS=1,                 (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=2,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=Y,                    FORCE UC: (YES, NO)            *
  KEYHDL=B,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  LNGKEY=8,                (NOT IN THE DA SENSE)          *
  TXTLNG=(1,92),           TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=(ASK,1),                                           *
  DIM4=(ASK,2)
```

```
GFT=360-FORTRAN-SOURCE,                                   *
  RECFM=FB,                (FB, VBA, ETC. ETC.)           *
  LRECL=80,                (80, 121, 137, ETC.)           *
  BLKSIZE=(80,4000),       ((MIN, MAX) OR FIXED)          *
  SPACE=(005,01),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=40,               (EST IL/DISK BYTES * 100)      *
  KOFFS=73,                (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=7,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=Y,                    FORCE UC: (YES, NO)            *
  KEYHDL=B,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  LNGKEY=8,                (NOT IN THE DA SENSE)          *
  TXTLNG=72,               TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=(ASK,1),                                           *
  DIM4=(ASK,2)


GFT=360-LIST,                                             *
  RECFM=VB,                (FB, VBA, ETC. ETC.)           *
  LRECL=136,               (80, 121, 137, ETC.)           *
  BLKSIZE=(140,4000),      ((MIN, MAX) OR FIXED)          *
  SPACE=(010,02),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=50,               (EST IL/DISK BYTES * 100)      *
  KOFFS=0,                 (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=1,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=N,                    FORCE UC: (YES, NO)            *
  KEYHDL=N,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=F,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  LNGKEY=0,                (NOT IN THE DA SENSE)          *
  TXTLNG=(1,132),          TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=(PERMIT,1),                                        *
  DIM3=(PERMIT,2),                                        *
  DIM4=(ASK,3)
```

```
GFT=360-CARDS,                                            *
  RECFM=FB,                (FB, VBA, ETC. ETC.)           *
  LRECL=80,                (80, 121, 137, ETC.)           *
  BLKSIZE=(80,4000),       ((MIN, MAX) OR FIXED)          *
  SPACE=(20,40),           (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=40,               (EST IL/DISK BYTES * 100)      *
  FOLDMGN=1,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=N,                    FORCE UC: (YES, NO)            *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  TXTLNG=80,               TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=(ASK,1),                                           *
  DIM4=(ASK,2)


GFT=360-OBJECT,                                           *
  RECFM=FB,                (FB, VBA, ETC. ETC.)           *
  LRECL=80,                (80, 121, 137, ETC.)           *
  BLKSIZE=(80,3200),       ((MIN, MAX) OR FIXED)          *
  SPACE=(20,40),           (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=40,               (EST IL/DISK BYTES * 100)      *
  KOFFS=73,                (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=1,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=N,                    FORCE UC: (YES, NO)            *
  KEYHDL=G,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=B,                 (TEXT, BINARY)                 *
  KPFILL=Y,                (TRAIL FILLS ARE SIGNIF; Y, N) *
  SUPRXL=Y,                (PERFORM CODE TRANS; Y, N)     *
  LNGKEY=8,                (NOT IN THE DA SENSE)          *
  TXTLNG=72,               TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=(ASK,1),                                           *
  DIM4=(ASK,2)
```

```
GFT=360-OVERPRINT,                                        *
  RECFM=VBA,               (FB, VBA, ETC. ETC.)           *
  LRECL=137,               (80, 121, 137, ETC.)           *
  BLKSIZE=(141,4000),      ((MIN, MAX) OR FIXED)          *
  SPACE=(010,02),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=50,               (EST IL/DISK BYTES * 100)      *
  KOFFS=0,                 (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=1,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=4,                   (NATIVE DIMENSION)             *
  UC=N,                    FORCE UC: (YES, NO)            *
  KEYHDL=N,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=F,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  LNGKEY=0,                (NOT IN THE DA SENSE)          *
  TXTLNG=(1,132),          TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=(PERMIT,3),                                        *
  DIM3=(PERMIT,2),                                        *
  DIM4=(PERMIT,1)


GFT=360-ASM-SOURCE,                                       *
  RECFM=FB,                (FB, VBA, ETC. ETC.)           *
  LRECL=80,                (80, 121, 137, ETC.)           *
  BLKSIZE=(80,4000),       ((MIN, MAX) OR FIXED)          *
  SPACE=(005,01),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=40,               (EST IL/DISK BYTES * 100)      *
  KOFFS=73,                (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=16,              (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=Y,                    FORCE UC: (YES, NO)            *
  KEYHDL=G,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  LNGKEY=8,                (NOT IN THE DA SENSE)          *
  TXTLNG=72,               TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=(ASK,1),                                           *
  DIM4=(ASK,2)
```

```
GFT=360-COBOL-SOURCE,                                     *
  RECFM=FB,                (FB, VBA, ETC. ETC.)           *
  LRECL=80,                (80, 121, 137, ETC.)           *
  BLKSIZE=(80,4000),       ((MIN, MAX) OR FIXED)          *
  SPACE=(005,01),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=40,               (EST IL/DISK BYTES * 100)      *
  KOFFS=1,                 (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=6,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=Y,                    FORCE UC: (YES, NO)            *
  KEYHDL=G,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  LNGKEY=6,                (NOT IN THE DA SENSE)          *
  TXTLNG=74,               TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=(ASK,1),                                           *
  DIM4=(ASK,2)


GFT=360-COBOL-SEQ-SOURCE,                                 *
  RECFM=FB,                (FB, VBA, ETC. ETC.)           *
  LRECL=80,                (80, 121, 137, ETC.)           *
  BLKSIZE=(80,4000),       ((MIN, MAX) OR FIXED)          *
  SPACE=(005,01),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=40,               (EST IL/DISK BYTES * 100)      *
  KOFFS=73,                (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=12,              (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=Y,                    FORCE UC: (YES, NO)            *
  KEYHDL=G,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  LNGKEY=8,                (NOT IN THE DA SENSE)          *
  TXTLNG=72,               TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=(ASK,1),                                           *
  DIM4=(ASK,2)
```

```
GFT=360-ORIGINAL,                                         *
  RECFM=VBA,               (FB, VBA, ETC. ETC.)           *
  LRECL=1000,              (80, 121, 137, ETC.)           *
  BLKSIZE=(1004,4000),     ((MIN, MAX) OR FIXED)          *
  SPACE=(010,02),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=100,              (EST IL/DISK BYTES * 100)      *
  KOFFS=0,                 (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=1,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=4,                   (NATIVE DIMENSION)             *
  UC=N,                    FORCE UC: (YES, NO)            *
  KEYHDL=N,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=F,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=Y,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  LNGKEY=0,                (NOT IN THE DA SENSE)          *
  TXTLNG=(1,1000),         TEXT LENGTH RANGE              *
  DIM1=PERMIT,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=PERMIT,                                            *
  DIM4=PERMIT


GFT=360-ORIGINAL-BIN,                                ******
  RECFM=VBA,               (FB, VBA, ETC. ETC.)           *
  LRECL=1000,              (80, 121, 137, ETC.)           *
  BLKSIZE=(1004,4000),     ((MIN, MAX) OR FIXED)          *
  SPACE=(010,02),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=100,              (EST IL/DISK BYTES * 100)      *
  KOFFS=0,                 (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=1,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=4,                   (NATIVE DIMENSION)             *
  UC=N,                    FORCE UC: (YES, NO)            *
  KEYHDL=N,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=F,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=Y,                (YES, NO)                      *
  CLASS=B,                 (TEXT, BINARY)                 *
  LNGKEY=0,                (NOT IN THE DA SENSE)          *
  TXTLNG=(1,1000),         TEXT LENGTH RANGE              *
  DIM1=PERMIT,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=PERMIT,                                            *
  DIM4=PERMIT
```

```
GFT=360-ASM80-SOURCE,                                     *
  RECFM=FB,                (FB, VBA, ETC. ETC.)           *
  LRECL=80,                (80, 121, 137, ETC.)           *
  BLKSIZE=(80,4000),       ((MIN, MAX) OR FIXED)          *
  SPACE=(005,01),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=40,               (EST IL/DISK BYTES * 100)      *
  KOFFS=73,                (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=16,              (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=Y,                    FORCE UC: (YES, NO)            *
  KEYHDL=G,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  LNGKEY=8,                (NOT IN THE DA SENSE)          *
  TXTLNG=72,               TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=(ASK,1),                                           *
  DIM4=(ASK,2)


GFT=360-CMS2M-SOURCE,                                     *
  RECFM=FB,                (FB, VBA, ETC. ETC.)           *
  LRECL=80,                (80, 121, 137, ETC.)           *
  BLKSIZE=(80,4000),       ((MIN, MAX) OR FIXED)          *
  SPACE=(005,01),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=40,               (EST IL/DISK BYTES * 100)      *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=Y,                    FORCE UC: (YES, NO)            *
  KEYHDL=B,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=(ASK,1),                                           *
  DIM4=(ASK,2),   *******FOLLOWING ARE TEMPORARY **********
  KOFFS=0,      (1)        (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=7,    (1)        (1 ORG, IN TEXT FIELD ONLY)    *
  LNGKEY=0,     (6)        (NOT IN THE DA SENSE)          *
  TXTLNG=80     (74)       TEXT LENGTH RANGE
```

```
GFT=360-PLM80-SOURCE,                                     *
  RECFM=FB,                (FB, VBA, ETC. ETC.)           *
  LRECL=80,                (80, 121, 137, ETC.)           *
  BLKSIZE=(80,4000),       ((MIN, MAX) OR FIXED)          *
  SPACE=(005,01),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=40,               (EST IL/DISK BYTES * 100)      *
  KOFFS=73,                (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=1,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=Y,                    FORCE UC: (YES, NO)            *
  KEYHDL=B,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  LNGKEY=8,                (NOT IN THE DA SENSE)          *
  TXTLNG=72,               TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=(ASK,1),                                           *
  DIM4=(ASK,2)
```

```
GFT=360-SPPCOBOL-SOURCE,                                  *
  RECFM=FB,                (FB, VBA, ETC. ETC.)           *
  LRECL=80,                (80, 121, 137, ETC.)           *
  BLKSIZE=(80,4000),       ((MIN, MAX) OR FIXED)          *
  SPACE=(005,01),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=40,               (EST IL/DISK BYTES * 100)      *
  KOFFS=1,                 (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=6,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=Y,                    FORCE UC: (YES, NO)            *
  KEYHDL=G,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  LNGKEY=6,                (NOT IN THE DA SENSE)          *
  TXTLNG=74,               TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=(ASK,1),                                           *
  DIM4=(ASK,2)
```

```
GFT=360-CMS2M-OBJ,                                        *
  RECFM=VBS,               (FB, VBA, ETC. ETC.)           *
  LRECL=3516,              (80, 121, 137, ETC.)           *
  BLKSIZE=(3520,3520),     ((MIN, MAX) OR FIXED)          *
  SPACE=(20,40),           (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=40,               (EST IL/DISK BYTES * 100)      *
  FOLDMGN=1,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=N,                    FORCE UC: (YES, NO)            *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=B,                 (TEXT, BINARY)                 *
  KPFILL=Y,                (TRAIL FILLS ARE SIGNIF; Y, N) *
  SUPRXL=Y,                (PERFORM CODE TRANS; Y, N)     *
  LNGKEY=0,                (NOT IN THE DA SENSE)          *
  TXTLNG=3512,             TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=(ASK,1),                                           *
  DIM4=(ASK,2)


GFT=360-PLM80-OBJ,                                        *
  RECFM=FBA,               (FB, VBA, ETC. ETC.)           *
  LRECL=133,               (80, 121, 137, ETC.)           *
  BLKSIZE=(133,3500),      ((MIN, MAX) OR FIXED)          *
  SPACE=(20,40),           (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=40,               (EST IL/DISK BYTES * 100)      *
  FOLDMGN=1,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=3,                   (NATIVE DIMENSION)             *
  UC=N,                    FORCE UC: (YES, NO)            *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  KPFILL=Y,                (TRAIL FILLS ARE SIGNIF; Y, N) *
  SUPRXL=Y,                (PERFORM CODE TRANS; Y, N)     *
  LNGKEY=0,                (NOT IN THE DA SENSE)          *
  TXTLNG=133,              TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                            *
  DIM3=PERMIT,                                            *
  DIM4=(ASK,2)
```

```
GFT=360-ASM80-OBJ,                                         *
  RECFM=FB,                 (FB, VBA, ETC. ETC.)           *
  LRECL=132,                (80, 121, 137, ETC.)           *
  BLKSIZE=(132,3500),       ((MIN, MAX) OR FIXED)          *
  SPACE=(20,40),            (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=40,                (EST IL/DISK BYTES * 100)      *
  FOLDMGN=1,                (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                    (NATIVE DIMENSION)             *
  UC=N,                     FORCE UC: (YES, NO)            *
  LONGR=F,                  (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                 (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                  (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                 (PAD, DON'T PAD; SHORT RECS)   *
  CLASS=T,                  (TEXT, BINARY)                 *
  KPFILL=Y,                 (TRAIL FILLS ARE SIGNIF; Y, N) *
  SUPRXL=Y,                 (PERFORM CODE TRANS; Y, N)     *
  LNGKEY=0,                 (NOT IN THE DA SENSE)          *
  TXTLNG=133,               TEXT LENGTH RANGE              *
  INONLY=N,                 (YES, NO)                      *
  DIM1=FORBID,              DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                             *
  DIM3=(ASK,1),                                            *
  DIM4=(ASK,2)
```

```
GFT=360-JCL,                                               *
  RECFM=FB,                 (FB, VBA, ETC. ETC.)           *
  LRECL=80,                 (80, 121, 137, ETC.)           *
  BLKSIZE=(80,4000),        ((MIN, MAX) OR FIXED)          *
  SPACE=(005,01),           (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=40,                (EST IL/DISK BYTES * 100)      *
  KOFFS=73,                 (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=16,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                    (NATIVE DIMENSION)             *
  UC=Y,                     FORCE UC: (YES, NO)            *
  KEYHDL=G,                 (REQ, GEN, BLANK, NO)          *
  LONGR=F,                  (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                 (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=A,                  (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                 (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                 (YES, NO)                      *
  CLASS=T,                  (TEXT, BINARY)                 *
  LNGKEY=8,                 (NOT IN THE DA SENSE)          *
  TXTLNG=72,                TEXT LENGTH RANGE              *
  DIM1=FORBID,              DIMENSIONAL PREFERENCES        *
  DIM2=PERMIT,                                             *
  DIM3=(ASK,1),                                            *
  DIM4=(ASK,2)
```

```
GFT=360-TEXT,                                             *
  RECFM=VB,                (FB, VBA, ETC. ETC.)           *
  LRECL=136,               (80, 121, 137, ETC.)           *
  BLKSIZE=(140,4000),      ((MIN, MAX) OR FIXED)          *
  SPACE=(010,02),          (PRIMARY, SECONDARY, DIRECTORY)*
  CMPFAC=50,               (EST IL/DISK BYTES * 100)      *
  KOFFS=0,                 (1-ORG, IN TEXT FIELD ONLY)    *
  FOLDMGN=1,               (1 ORG, IN TEXT FIELD ONLY)    *
  DIM=2,                   (NATIVE DIMENSION)             *
  UC=N,                    FORCE UC: (YES, NO)            *
  KEYHDL=N,                (REQ, GEN, BLANK, NO)          *
  LONGR=F,                 (TRUNCATE, FOLD; LONG RECS)    *
  SHORTR=P,                (PAD, DON'T PAD; SHORT RECS)   *
  LONGP=F,                 (TRUNC, FOLD, ALLOW; L PAGES)  *
  SHORTP=D,                (PAD, DON'T PAD; SHORT RECS)   *
  INONLY=N,                (YES, NO)                      *
  CLASS=T,                 (TEXT, BINARY)                 *
  LNGKEY=0,                (NOT IN THE DA SENSE)          *
  TXTLNG=(1,132),          TEXT LENGTH RANGE              *
  DIM1=FORBID,             DIMENSIONAL PREFERENCES        *
  DIM2=(PERMIT,1),                                        *
  DIM3=(PERMIT,2),                                        *
  DIM4=(ASK,3)
```

REFERENCES

1) Cashman, Faneuf, and Muntz, "File Package: The File Handling Facility for the National Software Works". Document CADD-7612-2711, Massachusetts Computer Associates, Wakefield, Massachusetts, Revised December 27, 1976.

2) Braden and Ludlam, "FP/360 -- The NSW MVT File Package". UCLA/OAC document UCNSW-204, November 20, 1980.

3) Ludlam and Rivas, "PL/MSG -- An MSG Interface for PL/I". UCLA/OAC document UCNSW-401, November 15, 1980.

4) DeLa Roca and Ludlam, "PLIDAIR -- Dynamic Allocation from PL/I". UCLA/OAC document UCNSW-407, February 11, 1980.

5) Ludlam, "PL/PCP -- An NSW Procedure-Call Protocol Package for PL/I". UCLA/OAC document UCNSW-402, November 15, 1980.

PART IV

UCLA RECCOMMENDATIONS ON LIBRARIES IN NSW

This section is separately available
as UCLA document TR-16

TABLE OF CONTENTS

4. PART IV: UCLA RECCOMMENDATIONS ON LIBRARIES IN NSW

4.1. THE SPECIFIC PROBLEM

4.1.1. IBM TOOLS IN NSW

If NSW is to provide an environment for the installation of software tools written for a specific family of computer systems, it must provide an interface between these tools and the NSW filespace. Specifically, it must provide a mapping between file constructs that are used by the tools and those that can be provided by the NSW encapsulating foreman as implemented for that system family. If such a mapping is difficult to define, it may be that the NSW filespace design should be embellished accordingly.

If NSW is to support the large number of tools that have been written for the popular IBM System/360-370 family, it must provide a mapping between the NSW filespace and the IBM Basic Partitioned Access Method, or BPAM, since operation of almost all language processors written for these systems depends heavily on BPAM's use.

BPAM is an IBM program's interface to the IBM Partitioned Data Set (PDS) construct. The PDS, in turn, is IBM's implementation of the notion of a file library. There is no similar notion in the current specifications of the NSW filespace.

4.1.2. FILE LIBRARIES

Logically, a library is a data construct which defines a search scope. The objects being searched for carry names in a namespace internal to the library. This namespace may be private to the library and its users, and independent of the namespace within which the library itself is named, or it may be a private subspace of the larger namespace. Library files of some form have been with us since there were linking loaders. Most major computer language definitions include specification of some form of library facility. This is true of COBOL, of PL/I, and of macro-assemblers as a class. A library facility is a part of the DOD published requirements for ADA (nee DOD-1), as is the ability to structure that library along application, project, and user lines.

A library is like a file in that the library name is a name in the host operating system's file name space, and is what becomes bound to a program's file by the operating system's file-management facilities when the program intends to use that file as a search scope. In this case, the internal namespace is not known to the file-management system, and is fully managed by the program.

A library is like a collection of files in that the name formed by qualifying a name from its internal namespace by its file name is itself a legitimate file name, and can be bound to a program's file when that program intends to use that file as a sequential input source.

A library can thus serve as a convenient repository for a general collection of like files. This usage is an inessential side effect; however, it points out that, where a filespace is hierarchically structured, there is great similarity between a library and a filespace subtree.

4.1.3. THE IBM IMPLEMENTATION

4.1.3.1. DESCRIPTION

There are several operating systems for the IBM System/360-370 family, but NSW will probably be concerned with two: MVT, a real-memory system, and MVS, a virtual-memory system. In both systems, a library is structured as a PDS. This structure has been a part of IBM systems since OS/360 was announced in 1964. In current systems, PDS's are not only supported, they are required, as is illustrated by the fact that one of the supervisor's basic services, program fetch, demands a PDS.

It is a good guess that future IBM system announcements will continue to require PDS's, and it is a certainty that they will continue to support them. At any rate, should IBM ever cease PDS support, systems using PDS's will still be in the field for some years thereafter.

A PDS is a collection of named files of like type, structure, and function, with a single file name, and a single disk allocation and attribute set. Each "member" of the collection has a simple name, and may have a number of "aliases". The set of names and aliases are tabulated in a special "member" called the "directory". The directory, by virtue of being unnamed, is not normally available to processing programs directly.

On the system command level (JCL or TSO), a reference to a PDS can name either the collection or a specific "member" of the collection. In the latter case, the reference is to a sequential file which can be processed as such by almost any program which uses the IBM sequential access method (SAM) file interface. With some unimportant exceptions, a reference to the PDS as a collection can only be processed by a program which consciously uses the BPAM interface, that is, one which operates on a library.

A PDS as a library is indicated whenever a program is to be told "whenever you need to locate a named piece of data of this certain sort, search this subset of the file space." This is particularly true when the name of the piece of data cannot be known until execution time. Common applications include: a command language executor searches a command library for the program named the same as the command just entered; a linking loader searches a subroutine library for routines whose names match the unresolved subroutine references within a program; an assembler searches a macro-definition library whenever an otherwise undefined operation code is encountered; the PL/I compiler searches a text library whenever a "%INCLUDE" statement is processed.

IBM files are allocated in terms of contiguous extents of real disk space. When a member is replaced in a PDS, its space is not reused. This is nicely compatible with the notion of version numbers for members, since under that notion the old member is not to be destroyed in any case; however, it does mean that space must be garbage collected on occasion. To the local user, the garbage collection problem appears only as the necessity to run an occasional "compress" utility on the library. This is either done whenever the user feels it may be needed, or when an attempt to add a member fails. When the "user" is a file package, more concrete criteria will have to be defined. This is essentially a local file package implementor's problem.

4.1.3.2. TYPICAL USAGE

Meaningful use of a library facility requires a "concatenation" facility, that is, a method to allocate an ordered set of libraries to a program in order to define a search rule. NSW support for concatenated allocation is not now provided, but it will not be difficult, and it need not be discussed here.

The number of members and volume of data in a PDS vary drastically according to the application. As an example of typical data volumes, consider the concatenation of PDS's that is routinely allocated to the OS/360 assembler's macro library file in order to assemble a file package routine:

IBM macro library:	313 members,	4.0 MB
UCLA macro library:	812 members,	4.9 MB
NSW macro library:	23 members,	115 KB
FLPKG macro library:	58 members,	524 KB
TOTALS:	1206 MEMBERS,	9.5 MB

A typical file package assembly will actually select and read perhaps 100 kilobytes, or about a dozen members of this data. Notice that not only is a rich variety of macro definitions made available to the programmer, but a search rule is specified by the order of concatenation. The user must be given control of his search order, so he must be able to specify an ordered set of NSW file names to be provided to a tool.

This presents a new naming problem. If the tool user is be able to name the IBM-provided macro library at UCLA, then that library must have an NSW file name. However, the local name of that library is fixed, and cannot be chosen by the file package in the same way that a true NSW file name is. Therefore, a mechanism must be provided for assigning an existing local name to a new NSW file name. A slightly expanded version of FP-IMP could do this.

A consequence of the dual nature of PDS access is that a PDS may be used, at the programmer's discretion, to hold any collection of similar, related files, whether or not they are ever to be accessed as a library, and to reference and operate upon them collectively.

4.1.3.3. MISCELLANEOUS PROBLEMS

Other problems arise when one tries to address PDS's and their members via NSW. Some of these include:

* Version numbers need to be provided for members instead of for libraries.

* Member names may need to be both mapped and unmapped in the NSW file name space.

* Access to a PDS member can make the entire library appear busy to the host operating system.

* Physical copies of the same NSW file must be capable of existing both as sequential files and as members of PDS's.

* True encapsulation of BPAM presents political difficulties that may make it impractical.

* Alias management will surely cause complications somewhere.

4.1.4. DESIDERATA

We feel that the absence of any form of library support in NSW is a serious omission. The entire NSW community could benefit from the integration of the library concept directly into the NSW file structure. Direct benefits could include:

* IBM tools could be supported.

* The IBM library implementation could be enriched by merging in good NSW concepts like version numbers for members. (Note that NSW file version number support is being implemented in NSW.)

* A mechanism for defining search scopes in NSW would be provided. This would be especially important for "new" (unencapsulated) tools on non-IBM hosts.

* A convenient and efficient form of collective file operations would become available to NSW users.

* Installation-provided and maintained libraries could be shared with NSW users.

4.2. THE PROBLEM IN PERSPECTIVE

4.2.1. THE GENERAL PROBLEM OF UNMOVEABLE FILES

The library problem in NSW is related to the general problem of NSW-unmoveable file types. These include all non-sequential types, such as those processed by IBM's Basic Direct Access Method (BDAM), Indexed Sequential Access Method (ISAM), or Virtual Storage Access Method (VSAM). They include files with an affinity to a particular tool, host, or operating system, as a database has affinity to a DBMS implementation. They also include files which, by virtue of their great mass, are not practically moveable. If a library is represented as a file, it is usually unmoveable due to its great mass. If it is represented as a PDS, it is also unmoveable, except for "family copies", due to its non-sequential nature and its affinity for an operating system.

It is likely that most unmoveable files will be integrated into NSW through a simplistic model in which the construct is given an NSW name, perhaps even an NSW file name, but it is operated on only by tool code, never by NSW code per se.

4.2.2. THE GENERAL PROBLEM OF COLLECTIVE FILES

The library problem is also related to the general problem of collective file types. These also include "stacked tapes". "Stacked tapes" will be important in NSW because they frequently will be the exported results when NSW is used for cross-system program development for systems such as the UYK-20. They consist of a sequence of sequential files, any of which could be processed as a legitimate NSW file. Once "stacked" as a "load tape" image, these will be collectively known by one NSW file name, and that collective file must be transmittable by the NSW to a host capable of writing a UYK-20 load tape. The NSW file package's "IL" data representation has been given a special "end-of-subfile" construct to accomodate this transmission, although it is not yet being used. While this usage does not require that the individual members of the "stack" be independently retrievable and replaceable, that usage could develop. In either case, it is highly probable that we will represent a "stacked tape" on the IBM system as a PDS.

Another type of collective file that will see use in NSW is the "mail file" of the TENEX sort. A mail file is really a special use of a library file, and an IBM implementation could well use a PDS.

4.3. PROPOSED SOLUTIONS

4.3.1. THE DATABASE MODEL

4.3.1.1. DESCRIPTION

In this model, a PDS is considered to be a special form of database -- it has an NSW file name, but not an NSW file structure. Its NSW file name can be allocated to any batch or interactive tool, provided that tool resides on the same host as the single physical copy of the file. The IBM file package will never make a tool copy of such a file, but will always grant the tool access to the NSW copy directly (this mechanism is already implemented in the IBM file package). Thus encapsulated tools can access such a file in native mode; however, explicit tools must be provided for file maintenance.

In the general database model, file structure and implementation are dictated by a tool or set of tools, and maintenance functions are included in that set, since the set of meaningful maintenance functions depends on the file structure and implementation. In the case of PDS maintenance, the following probably represents a minimal set of maintenance functions to be provided by a special tool or tools:

1) Create a PDS.

2) Copy an NSW file into a PDS member.

3) Copy a PDS member into an NSW file.

4) List the member and alias names.

5) Delete a member or alias.

6) Rename a member or alias.

7) Assign an alias to a member.

8) Analyze a PDS for garbage content.

9) Compress (garbage collect) the PDS.

This model would enable any IBM tool to process a PDS as a library, but not to process a member as a sequential file, since the member has no NSW file name. However, any NSW tool, IBM or otherwise, could operate on a member's data if the user made explicit use of the maintenance tools to extract and replace it. Specific IBM tools could regain their ability to operate on a PDS member by using a local NSW-provided preface routine, such as the UCLA Encapsulator Command Interpretor (ECI) to explicitly refine the PDS allocation into a member allocation. This mechanism would be supplied initially to the IBM TSO EDIT tool, since it accounts for the great majority of sequential references to PDS members.

In order to bring a PDS unchanged into NSW, a special version of IMPORT would be required. This version must support assigning an NSW file name to a given existing file which NSW will share with non-NSW users of the host.

4.3.1.2. EVALUATION

The database model is simple and easy to implement; however, it fills only one NSW desideratum. It makes it possible to integrate IBM tools into NSW, but at the expense of using non-NSW methods and mechanisms to do a large part of the kinds of processing usually associated with program development tasks. It does not improve upon the library functions available from the IBM implementation, nor does it provide search scopes and collective operations to non-IBM tools or to NSW users. It makes library data available for general use only through special user action. It violates a fundamental NSW precept by allowing a tool write access to the NSW copy of a file.

Nevertheless, the database model has one overpowering advantage -- it can be implemented by host software development personnel without requiring any changes at all to the works manager or to other NSW code. The probable result of this fact is that, whatever model is selected for integrating libraries into NSW properly, the database model is going to be used in the interim to support existing needs. Such interim support will be fully sufficient for existing tools and for those planned for the immersion project ("Immersion" is used to denote the adoption of the NSW as the sole or primary vehicle for the development, maintenance, and configuration of the NSW itself).

4.3.2. THE NSW SCOPE MODEL

4.3.2.1. DESCRIPTION

This model is based upon the similarity between a library and a subtree of a hierarchically structured filespace. NSW has such a filespace, and calls such a subtree a scope. The scope model hinges on these features:

* NSW must allow a user to allocate an NSW scope name instead of an NSW file name to a tool.

* BPAM must be encapsulated to the point of giving control to NSW code whenever a tool issues the supervisor service call that locates members within a PDS (the BLDL SVC call).

* The NSW IBM encapsulating foreman must form filespecs by qualifying each requested member name by the allocated scope name, must issue WM-GET calls against these filespecs, and must copy any resulting files into a true PDS in the tool's workspace, under the appropriate member names.

* The information flow of WM-GET must be embellished to communicate from the foreman to the file package the name of the PDS into which the tool copy is to be made.

* BPAM must also be encapsulated to the point of giving control to NSW code whenever a tool issues the supervisor service call that creates members within a PDS (the STOW SVC call).

* The NSW IBM encapsulating foreman must retain the name of each newly-created member in its LND, as each constitutes a deliverable file.

* The NSW user interface, or at least the IBS component, must be sophisticated to manage the case where the set of deliverables for a tool instance is bound after tool execution.

4.3.2.2. EVALUATION

The scope model has much appeal. It integrates IBM tools into NSW, and it does it through the encapsulation technique that NSW prefers for such purposes. It does embellish upon the PDS implementation such that "members", being NSW files, can have all the attrictive NSW properties, like version numbers. It provides mechanisms that will be useful to "new" tools on non-IBM hosts.

On the negative side, the scope model can be expected to require an order of magnitude more space in the NSW file catalog, since not only are all libraries there, but each "member" is individually represented by an NSW file name. This model does not provide more efficient collective operations on libraries, since NSW file operations on scopes will be simple iterations of file operations.

Since local TBH libraries must still be represented as PDS's, this model can only handle user data. This will prevent the user's specifying a search order between public and private libraries.

But the crippling disadvantages of the scope model are matters of practicality and efficiency. Such an implementation is not feasible under a real-memory batch-processing system such as OS/360 MVT, due to various combinations of the following facts:

* All network I/O for a real-memory batch job MUST be prestaged. Otherwise, a typical two-minute assembly could be stretched to half an hour, with one batch stream and perhaps 200K of storage tied up for that interval. This is because a typical assembly will ask for a dozen macros, one at a time, when it decides it needs them. Each would be a WM-GET request, and could take several minutes.

* It is not sensible to prestage an entire search scope, when only a very small percentage of the data will actually be read by the tool.

* It is not possible to predict which parts of a search scope will be needed.

* Even if the entire search scope could be coaxed into staying on one host, the encapsulation of the elementary supervisor services used by IBM tools to search and read libraries is not politically practical. Without system modifications, the search scope MUST be structured as a concatenation of Partitioned Data Sets. The modifications to the IBM supervisor to make this not so are not likely to be allowed by any systems manager.

It may be that some of these problems do not apply to MVS. Under a virtual-memory system it may be practical to do no prestaging of library data at all, since waiting for a WM-GET will not tie up valuable hardware. However, remember that the only IBM installation currently in NSW is an MVT batch system, and that significant numbers of such systems exist in DOD installations.

Unfortunately, neither MVT nor MVS provides a general mechanism comparable to the JSYS traps of TENEX, so the BLDL and STOW routines must be modified in order to encapsulate those BPAM functions. Such modifications are not to be done lightly, as they cause a loss of IBM software support, so that many IBM installations would not consider them. Unlike some vendors, IBM is quite unresponsive to requests that they add such changes as supported features.

4.3.3. THE COLLECTIVE FILE MODEL -- BASIC FORM

4.3.3.1. DESCRIPTION

In this model, the NSW filespace is embellished to include the notion of a file library construct, but not the notion of version numbers for library members. Library representation is presumed to be host-family specific, so each host family is allowed to specify its own implementation. Libraries thus become more or less unmoveable; however, library members can move freely.

The basic collective file model hinges on these features:

* The works manager is aware of two new file attributes, "library" and "member".

* NSW must allow a user to allocate libraries and members, as well as ordinary NSW files, to a tool.

* An NSW file with the "library" attribute is always so represented in the NSW file catalog. It may not have more than one physical copy. Otherwise, it is treated as a normal NSW file. There are some file operations to which it is not a legal argument, and the works manager could do some error checking; however, actual enforcement of any such restrictions will be the file package's responsibility.

* An NSW file acquires the library attribute when it is imported or delivered into a non-existent NSW file name. The importing file package reports the attribute to the works manager, who records it.

* An NSW filename or filespec is recognized to have the member attribute because of its syntactic form -- it consists of the qualification of a simple name by a legitimate NSW filename or filespec, using a unique qualification syntax.

* The NSW member name is the same as the local member name, but the library name is mapped as for any NSW file.

* A "member" file is represented explicitly in the NSW file catalog if and only if there is one or more physical copies other than the one in the library itself. The works manager will assume the existence of a member not so represented if the qualifying file name exists and has the library attribute. The file package has final responsibility for detecting the non-existence of a member name.

* When a member is exported from a library and a NSW copy is kept, an explicit NSW file catalog entry is made for the fully-qualified member name, and the new physical copy is entered into that entry. The presence of the member attribute implies the existence of the physical copy in the library itself.

* When there is an entry for a member, it is implicitly linked to the entry for its library through name similarity. This link can be found through existing disambiguation mechanisms.

* When a file is delivered into a "member" file, the corresponding library must already exist, and delivery must be performed by the file package at that site. The works manager must thus route the file package call accordingly. Presumably, all existing file package implementations have cross-network import capability.

* In general, when a file is delivered into an existing version of an NSW file, all physical copies of that file must be deleted. Specifically, when delivery is to a member, if that member name is explicitly represented in the file catalog, all its non-PDS physical copies are scheduled for deletion, and the member catalog entry is deleted. (The PDS copy will be deleted by the "replace" option on the file package call.)

* All file package calls must be embellished to include member names where appropriate. It will probably be wise to communicate the "library" and "member" bits somewhere in such calls, although the file package may be able to discover this for itself.

* FP-IMP must be able to report the library attribute in its reply. It must also have an option that assigns an NSW file name to an existing file which NSW will share with non-NSW users.

* When WM-GET is issued against a library name, the file package will grant access directly to the single NSW copy. The foreman must ensure read-only access. A request for write access to an entire library will be denied by the foreman.

* Delivery into an existing library name is not defined.

* A request to delete a member is not a special case, except that two file catalog entries may be involved.

* A request to rename a member must change only its member name. This request must be passed to the file package, as member names are not mapped by the works manager. Thus a new file package call must be defined.

* While this model does not provide versions for members, a library, being an NSW file, will have versions. The works manager must recognize that delivery into a member does not change the version number of the library itself.

* For the IBM implementation, a maintenance tool must be provided to handle at least these chores:

  1) Create a PDS.

  2) Assign/delete an alias.

  3) Analyze a PDS for garbage content.

  4) Compress a PDS.

4.3.3.2. EVALUATION

This model addresses more NSW desiderata than those previously described. IBM tools can be supported, and they can access PDS libraries or members without change. Non-IBM tools can access PDS members. A search scope construct is defined, and can be implemented in non-IBM file packages as well. Such implementations are not constrained to be similar to PDS's. Collective operations on libraries are possible. Non-NSW libraries can be named and used by NSW users. However, the PDS implementation itself is not embellished; specifically, version numbers are not provided for members.

This model does require extensive works manager modifications, but they are all straightforward. Only incremental space increases need be expected in the file catalog.

PDS garbage collection will have to be handled manually at first. It is possible that this can be automated later, but it is not necessary for implementation of the model.

4.3.4. THE COLLECTIVE FILE MODEL WITH VERSIONS

4.3.4.1. DESCRIPTION

This model is a direct extension of the previous one, and is only treated separately to avoid excessive descriptive complication. Additional features include:

* The works manager supports version numbers on simple member names. Version numbers may still exist for libraries, since they are NSW files, but they have less meaning now, and their use should probably be discouraged. It is possible that the works manager should specifically forbid version numbers for libraries.

* Member names are now mapped in a special way: the "current" version of a member has the same NSW name and local name; all other versions exist in the same library, but under generated names.

* The names for non-current members are chosen by the local file package in the same way that it chooses local names for physical copies of any NSW file. The internal namespace has an area fenced for this purpose.

* A member with no non-PDS physical copies and no non-current versions is not represented in the NSW file catalog. Non-PDS physical copies are represented as in the basic collective file model.

* Non-PDS physical copies of a non-current version are not permitted. When the current version number changes, any non-PDS copies are deleted.

* The file catalog entry for a member includes a mapping of version numbers and generated member names for all non-current versions. This list can have an NSW-wide maximum length beyond which old versions are deleted automatically. Normally, versions are deleted manually.

* When both the version number list and the physical copy list of a member entry in the file catalog become empty, the entry is deleted from the catalog.

* When delivery into a PDS is to a new version of an existing member, the file package must be told this. It must rename the old version to a generated name, store the new version under the member name, and report to the works manager the new name of the old version. The works manager must record this name in the version list of the member's entry in the file

catalog, creating such an entry if necessary.

4.3.4.2. EVALUATION

This model is indeed all things to all people. It fills all the library-related NSW desiderata, and significantly enhances the IBM PDS implementation. However, it requires even more code modification than the basic model.

Techniques for controlling version number growth would have to be explored before this model were implemented. Otherwise, it could cause a serious explosion in the file catalog. More serious for an IBM host system, PDS space could grow at an alarming rate.

4.4. RECOMMENDATIONS

NSW architects should consider all these models, along with hybrids of them, in designing a library facility for NSW. The feelings of the authors are these:

* Despite its attractive features, the NSW scope model has unfortunate deficiencies that make its implementation unattractive.

* The database model should be implemented at UCLA as soon as possible, to serve the needs of the immersion project and those of existing tools like MACRO80.

* MCA should plan an implementation based on the basic collective file model, and should try to have it ready before large numbers of users have to learn to use the interim database-model implementation.

* MCA should specify an extension to its implementation based on the collective file model with versions; however, implementation of this extension could be deferred for the time being.

* In any case, the notion of importing non-NSW files for shared use should be supported by NSW. Access to such files should always be read-only, and should be to the original, non-NSW copy. Maintenance should be outside NSW. This notion can be implemented and supported by individual file packages; however, it is useful enough to deserve a uniform, NSW-wide specification.